CENTRE DE RECHERCHE
SUR LES TRANSPORTS
(C.R.T.)
CENTRE FOR RESEARCH
ON TRANSPORTATION

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A,
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE: (514) 343-7575
TÉLEX: (21) 05-561325
EMAIL: crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX): (514) 343-7121

CRT-829

La recherche avec tabous et ses applications

par

Michel Toulouse

Centre de recherche sur les transports — Publication #829

Avril 1992

92-04612/

Dépôt légal – 2e trimestre 1992

Résumé

La recherche avec tabous est l'une des nouvelles approches qui sont apparues récemment pour la résolution de problèmes combinatoires. Cette méthode se base sur les données historiques de la recherche pour soustraire celle-ci de l'attraction des optimum locaux, du cyclage et des parcours sous-optimaux. Faisant usage de plusieurs principes de l'intelligence artificielle pour définir les règles d'évaluation qui dirigent la recherche, cette méthode se distingue par la flexibilité et la capacité d'adaptation des moyens pris pour trouver des ensembles intéressants de solutions. Le présent travail constitue un survol de cette méthode et de ses applications selon la perspective d'un informaticien plutôt que celle d'un chercheur en recherche opérationnelle. Conséquemment, une attention particulière est portée aux structures de données pour supporter l'implantation de cette méthode et à ses aspects reliés à l'intelligence artificielle. Également nous avons choisi de décrire certaines applications de la recherche avec tabous plus significatives du point de vue de l'informatique et de tenter de comprendre l'influence possible sur cette méthode de nouveaux modèles de calcul qui nous viennent de sciences comme la physique et la biologie.

Mots-clés: recherche avec tabous; optimisation; parallélisme; intelligence artificielle.

1 Introduction

Sous l'impulsion partielle des possibilités qu'offre la résolution de problèmes en parallèle, nous avons vu apparaître ou réapparaître récemment de nouvelles approches pour résoudre les problèmes combinatoires. On peut en citer quelques unes: les systèmes de règles de classification (classifier systems) Holland [31], les réseaux immunitaires (immune networks) Jerne [35], les réseaux autocatalytiques (autocatalytic networks) Farmer et Kauffman [8], les réseaux de neurones (neurals networks) McGulloch et Pitts [3] Rosenblatt [55] (ces 4 premières approches sont regroupées sous le vocable de connectionnisme [7]), les algorithmes génétiques Holland [31], le recuit simulé Kirpatrick [39], la recherche avec tabous de Glover [14].

Toutes ces approches ont en commun de faire appel, à des degrés divers, à une théorie du calcul basée sur l'apprentissage, et se distinguent sous cet aspect tout à fait des algorithmes traditionnels d'énumération comme le Branch-and-Bound, pour la résolution de problèmes complexe. En même temps que ces nouvelles approches, il y a le travail de nombreux physiciens qui a pour but de comprendre les phénomènes d'auto-régulation des systèmes parallèles d'ordinateurs. On assume de plus en plus que ces systèmes possèdent plusieurs des caractéristiques des organisations biologiques et sociales. Kephart, Hogg et Huberman [37] comparent les systèmes parallèles à des "organismes de calcul" composés d'agents prenant des décisions de façon autonome en réaction à un environnement. R. Maxion [49] pense qu'on peut utiliser la capacité d'adaptation collective à l'environnement des systèmes parallèles et Tad Hogg [38] discute de l'influence sur la performance d'un système parallèle si le comportement collectif est utilisé pour prévoir l'état futur du système.

Une théorie du calcul basée sur l'apprentissage et un développement de la recherche fortement contrôlé par des processus d'anticipation similaires à ceux utilisés en intelligence artificielle, voilà qui selon nous est un début pour décrire une méthode récente en progression rapide, la recherche avec tabous. Dans la section qui vient, nous allons faire une brève introduction de l'environnement en recherche opérationnelle qui a entouré la création et le développement de cette méthode, et nous ferons une mise en contexte pour le reste de ce travail. Les sections 3, 4 et 5 sont consacrés à la description du corps de la méthode des tabous. Il s'agit d'une description circulaire qui vise à introduire de façon graduelle le lecteur à tous les aspects qui nous sont parus essentiels ou révélateur de la philosophie de cette méthode. La section 6 concerne les développements de la recherche avec tabous pour les problèmes de grandes tailles. Dans la section 8 nous allons faire une présentation en détail de trois applications de la méthode à des problèmes combinatoires. La dernière section est une brève conclusion.

2 Contextes

Bon nombre des heuristiques pour résoudre les problèmes combinatoires sont des méthodes qu'on nomme de "Hill climbing". Ces heuristiques sont des processus qui, partant d'une solution initiale réalisable, modifient certains paramètres de la solution jusqu'à ce qu'il

n'existe plus aucune modification capable de provoquer un changement monotone de la valeur de la solution réalisable. Le processus s'arrête alors sur cette dernière solution qui n'est en général pas la meilleur solution possible. On dit alors que le processus a été bloqué par un optimum local.

Le but de la méthode de recherche avec tabous est d'empêcher que ces heuristiques ne restent bloquées dans des optimum locaux. Pour ce faire, cette méthode permet des changements aux paramètres de la solution qui peuvent faire varier la fonction objectif de façon non monotone. Ces changements sont contrôlés par des mécanismes qui empêchent la trajectoire de recherche de cycler. Pour cette raison, la recherche tabou est partout comparée à une méta-heuristique qui guide d'autres procédures ou heuristiques pour prévenir ces dernières d'être immobilisé par des solutions d'optimum local.

La méthode des tabous fut développée parallèlement par Glover [14], Hansen et Jaumard [27]. Dans l'esprit de Glover, cette méthode devait fournir un cadre qui permette la continuation de l'exploration de l'espace de solutions au-delà des points d'optimum locaux [18]. Elle était destinée à être utilisée pour diriger tout processus qui dispose d'un ensemble de mouvements alternatifs pour transformer une solution en une autre [15]. Comme exemple de mouvements alternatifs, on peut penser au *pivot* dans le simplexe ou encore aux *swaps* dans les problèmes de permutation des éléments d'un ensemble (permutation des arêtes dans le problème du voyageur de commerce). Nous utiliserons le terme *transformation* pour identifier ces mouvements.

Comme nous le mentionnons dans l'introduction de ce rapport, le progrès de la recherche avec tabous dans l'espace de solutions est fortement contrôlé par certaines inférences obtenues à partir de la trajectoire passée de la recherche. Glover exprime ce fait par une analogie entre les différentes composantes de sa méthode et les mémoires à court et long termes qui occupent différents espaces du cerveau humain [15, 16, 18, 24]. Le fonctionnement général de la méthode avec tabous mime en quelques sorte le rôle de ces différentes mémoires chez l'humain. C'est dans ce sens que lors de la description de la méthode avec tabous, nous parlerons parfois de fonctions de la mémoire à court, moyen et long terme.

Nous allons aussi faire notre propre analogie, cette fois dans le contexte de l'informatique. Le schéma global de la méthode apparaîtra alors comme une procédure formée de trois boucles imbriquées les unes dans les autres. La boucle interne constitue le corps de la procédure tandis que les deux autres bouclent contrôlent l'exécution des fonctions de la boucle interne. La plupart des fonctions de la procédure opèrent sur des données dont le degré de permanence en mémoire d'ordinateur décroit avec le degré de contrôle que ces fonctions exercent sur l'ensemble de la procédure. La boucle interne logera les fonctions dites de la mémoire à court terme de la méthode, la première boucle de contrôle celles de la mémoire à moyen terme, et la boucle externe celles de la mémoire à long terme.

3 Boucle interne: première version

Dans cette section, nous allons brièvement décrire les fonctions de la boucle interne qui permettent de réaliser l'essentiel de la recherche avec tabous, c'est-à-dire sortir des optimum locaux et contrôler le phénomène de cyclage.

Lorsqu'une heuristique de "Hill Climbing" se trouve dans un optimum local, toutes les solutions dans le voisinage de la solution optimale ne sont plus admissibles selon ces heuristiques. La méthode des tabous permet de poursuivre la recherche en ne tenant pas compte de ce critère d'arrêt. Cependant, cette relaxation des conditions de la recherche peut avoir sur une procédure de tabous un effet similaire aux solutions de base dégénérées dans le simplexe, elle peut provoquer l'apparition de trajectoires cycliques dans la recherche.

Les structures de données et les procédures qui sont contrôlées par la boucle interne ont d'abord pour fonction d'implanter les mécanismes qui permettent de sortir des optimum locaux, de contrôler le phénomène de cyclage et d'assurer un niveau de définition de la recherche qui soit suffisant pour ne pas laisser de poches inexplorées de l'espace de solutions.

Le problème des optimum locaux est éliminé facilement en ne tenant plus compte du critère d'arrêt de l'heuristique et en permettant l'application à la solution locale d'une transformation qui n'améliore pas nécessairement la fonction objectif.

Le contrôle du cyclage se fait à l'aide d'une trace qui est gardée en mémoire de la trajectoire récente de la recherche. La structure de données nécessaire pour conserver cette trace est un simple vecteur tl de taille t qu'on appelle *liste tabou*. La taille t de cette liste représente le nombre d'itérations successives pour lequel on veut garder la trace de la trajectoire.

Il y a différentes approches quant à savoir qu'est-ce qu'on garde dans tl pour représenter cette trajectoire. De Werra et Hertz [4] proposent de garder les t dernières solutions. Le fait qu'une solution se trouverait dans tl rendrait cette solution *tabou* et toute transformation qui produirait une solution de cette liste serait rejetée. Glover [18] propose de conserver un *attribut* des transformations, par exemple la variable pivot pour le pivot du simplexe, rendant tabou t transformations dont l'attribut se trouve dans la liste tabou. Cette dernière approche a l'avantage de permettre l'utilisation de plusieurs attributs en même temps pour chaque transformation. On peut ainsi déterminer empiriquement l'attribut ou la combinaison d'attributs qui est la plus efficace pour contrôler le cyclage, permettre à la recherche de trouver de meilleurs solutions et de progresser plus rapidement. Nous verrons plus loin d'autres variantes du tabou par attribut et les effets que ces différentes approches peuvent avoir sur la recherche.

Dans le contexte où il y a k listes tabous, on a comme structure de données k tableaux tl_j ($j \leq k$) de taille t_j tel que l'entrée $tl_j(i)$ contienne l'attribut j de la transformation e_i. Puisque chaque attribut reste dans sa liste tabou pour t itérations, une gestion statique[1] de ces listes tabous consiste à manipuler ces tableaux comme des listes circulaires introduisant l'attribut j de la transformation appliquée à l'itération i dans l'entrée $l = i \bmod t$ du tableau tl_j.

[1] Par rapport à la gestion dynamique que nous verrons dans une autre section.

Une transformation devient tabou dès qu'un de ses attributs se trouve dans une liste tabou, c'est-à-dire à l'application de la transformation à la solution courante. Elle reste tabou jusqu'à ce qu'aucun de ses attributs ne se trouvent dans une liste tabou. Ce n'est cependant pas la seule façon de lever le statut tabou d'une transformation. Il y a dans la méthode des tabous un mécanisme qui permet de lever le statut tabou d'une transformation même si certains attributs sont encore dans des listes tabous. Ce mécanisme existe pour empêcher que les listes tabous, par suite du statut tabou de certaines transformations, ne retardent l'orientation de la recherche dans une direction prometteuse et pour contrer le manque de définition que la recherche peut avoir sous l'effet des listes tabous.

Le mécanisme consiste de c *critères d'aspiration* qui servent à évaluer l'intérêt d'une solution obtenue à partir d'une transformation tabou. Le fonctionnement du critère d'aspiration est assez simple. Pour chaque critère c_i, on défini un seuil Θ_i tel que si l'évaluation de la solution par rapport à au moins un des critères est supérieure au seuil, alors le statut tabou est levé sur la transformation qui produit cette solution.

Les seuils que nous avons vus dans différentes applications de la recherche avec tabous sont souvent fonction des solutions trouvées dans les itérations antérieures, ils varient donc au cours de la progression de la recherche. En plus de la procédure qui décide de l'applicabilité des critères d'aspiration, il faut donc prévoir une procédure de mise à jour des seuils. Ces procédures pour l'application des critères d'aspiration et leur mise à jour ne représentent pas de façon générale un très grand effort de calcul.

Ceci termine un premier niveau de description du fonctionnement de la boucle interne d'un algorithme de la méthode des tabous. Cette description nous a permit d'identifier l'existence de procédures de gestion des listes tabous et de manipulation des critères d'aspiration. Ces procédures nous font voir sous sa forme la plus simple comment cette méthode utilise l'historique de la recherche pour diriger celle-ci en dehors des optimum locaux et prévenir le cyclage.

Ce que nous n'avons pas vu dans cette section ce sont les procédures qui permettent de décider de la meilleure transformation à appliquer pour chaque itération de la méthode. Bien que moins centrale dans le concept de la méthode, ces procédures sont le lieu où la méthode traite le phénomène d'explosion combinatoire. Le temps de calcul de ces procédures peut être considérable, aussi il y a là des choix qui ont un effet direct sur l'efficacité de la méthode. Cet aspect sera traitée en détail dans la section suivante de même que dans la section 5. Nous l'avons vu brièvement, il y a certaines considérations dont il faut tenir compte aux niveau des listes tabous et là aussi il y a des choix qui ne sont pas sans conséquence quant à l'efficacité de la recherche. Nous allons voir ces choix dans la prochaine section.

4 Boucle interne: deuxième version

Pour illustrer la séquence complète des opérations de la mémoire à court terme, nous allons décrire une application de la recherche avec tabous sur le problème du voyageur de commerce (TSP).

Nous allons supposer un exemplaire d'assez grande taille du problème du voyageur de commerce. Une *solution* au TSP sera tout *parcours cyclique* qui visite exactement une fois chaque sommet du graphe connexe (cycle Hamiltonien). Comme solution initiale, on choisi de façon aléatoire un tel parcours cyclique (pour abréger nous utiliserons le terme parcours à la place de parcours cyclique). Le mouvement de transformation sera le *swap 2-opt* qui consiste à soustraire du parcours deux arêtes non-adjacentes et à y ajouter deux autres arêtes du graphe qui constitueront un parcours différent.

Il est possible d'associer plusieurs attributs à une transformation comme le swap 2-opt pour le TSP. On peut utiliser le quadruplet d'arêtes qui entrent et sortent du parcours à chaque swap, où encore uniquement la paire d'arêtes qui sort du parcours, ou celle qui entre.

Le choix de la base pour décider si une transformation est tabou ou non est un élément clef pour les performances de la méthode des tabous. Avant de poursuivre, nous allons voir plus en détail bon nombre des considérations qui guident ce choix.

Nous allons supposer le modèle formel suivant d'un problème d'optimisation combinatoire. Soit P un problème de minimisation tel que

$$(P) \quad \text{Minimiser } c(x) : x \in X,\ X \subset Z^n$$

Une transformation e est une application définie sur le sous-ensemble $X(e)$ de X tel que

$$e = X(e) \longrightarrow X.$$

En association avec $x \in X$ on a l'ensemble $E(x)$ qui représente les transformations $e \in E$ qui peuvent être appliquées à x, d'où

$$E(x) = \{e \in E : x \in X(e)\}$$

et

$$X(e) = \{x \in X : e \in E(x)\}$$

Soit $x_i, x_j, ..., x_p$ une séquence de solutions tel que $x_j = e_i(x_i)$. Comme nous l'avons vu, l'objectif de rendre tabou certaines transformations est d'empêcher la répétition de séquences de solutions. Supposons que nous utilisions comme base la méthode de De Werra et Hertz [4] et que nous interdisions la répétition de toute solution dans la séquence $S = \{x_1, x_2, ..., x_k\}$. Soit x_q une solution telle que $x_q = e_i(x_i)$ où $x_i \in S$. On dit que $x_q \notin S$ est une solution dominée par x_i et la trajectoire de recherche qui passe par x_i, x_q est dominée par la trajectoire $\{x_1, x_2, ..., x_k\}$. Si on empêche la répétition de séquences de

solutions en gardant tabou toutes les solutions de S, on empêche également les séquences de solutions qui correspondent à des trajectoires dominées par S. Ceci a un effet direct sur la qualité de la solution qui résulterait d'une telle recherche, il y a de bonnes chances pour que cette solution soit assez éloignée de la solution optimale. Comme le soulignent De Werra et Hertz [4], une telle restriction sur le développement de la recherche n'est pas souhaitable, d'où l'utilité de limiter la taille des listes tabous.

Les listes tabou de taille limitées apparaîssent être empiriquement un bon moyen pour empêcher le cyclage, mais théoriquement on ne pourrait pas démontrer que le cyclage soit impossible. On voit donc un dilemme surgir. Plus t est grand plus on prémunit la recherche contre les trajectoires cycliques mais c'est au prix de laisser des zones inexplorées. Il y a un compromis entre le risque de blocage dans une région sous-optimale et celui de passer à côté de très bonnes solutions. Glover à d'abord pensé que 7 était la taille *magique* pour les listes *tabous* [18]. Mais les récentes applications de la recherche avec tabous [18, 44, 46] tendent à démontrer que t varie en fonction de la taille du problème et que même à l'intérieur de l'exécution d'un algorithme de tabou, la variation de t est parfois payante [56].

La base choisie pour décider du statut tabou est un autre paramètre auquel la taille de t est très sensible. D'après Glover [17] de même que Malek et Guruswamy [46], le choix des attributs et l'utilisation qu'on en fait pour rendre une transformation tabou peuvent exercer une contrainte plus ou moins forte sur la trajectoire de la recherche (en terme du nombre de transformations admissibles à chaque itération). Par exemple, l'utilisation d'attributs pour empêcher la répétition des transformations serait selon Glover moins contraignante qu'une utilisation qui vise à empêcher l'inversion des transformations. Dans ce contexte, l'utilisation d'attributs plus contraignants doit être contre-balancé par une taille plus petite de t et l'inverse lorsque les attributs ont un effet peu contraignant sur la recherche.

Glover propose l'utilisation de deux listes tabous pour des problèmes comme le TSP, l'une pour empêcher la répétition et l'autre l'inversion de transformations. Il nous semble quant à nous, qu'une bonne implantation de la méthode des tabous devrait avoir en paramètre le choix de plusieurs attributs pour permettre par l'expérimentation de définir la taille de chaque liste tabou en fonction de l'attribut ou de la combinaison d'attributs le plus efficace pour la recherche.

Si on revient au choix des trois attributs que nous avions pour le problème du TSP, nous choisirons le quadruplet comme attribut à placer dans les listes tabous. L'implantation de la liste tabou prendra la forme d'un tableau de $t \times 8$ où chaque arête est identifiée par une paire de noeuds connexes du graphe. Nous insérerons dans la liste tabou les paires de sommets $\{x, y\}$ et $\{w, z\}$ pour respectivement la paire d'arêtes $\{x, y\}$ et $\{w, z\}$ qui entre et sort du parcours par la transformation e. Nous empêchons donc la répétition des transformations.

Comme nous le mentionnions plus haut, il n'est pas souhaitable de se priver de trajectoires non-cycliques qui passent par une solution déjà visitée. Mais même une taille limitée des listes tabous peut faire en sorte que trop de solutions intéressantes soient inaccessibles. Il faut donc intervenir avec les critères d'aspiration[2]. Nous prendrons comme seul critère

[2]Il nous semble qu'à l'origine cette expression de "critère d'aspiration" devait faire référence à une

d'aspiration la valeur de la fonction objectif. Le seuil sera la valeur du meilleur parcours obtenu jusqu'à l'itération courante.

La grande taille de l'exemplaire du TSP implique, pour chaque itération, un grand nombre de solutions dans le voisinage de la solution courante x et conséquemment la cardinalité de $|E(x)|$ est grande. Nous identifions par ***transformations disponibles*** l'ensemble $E(x)$. Une des premières tâches de la boucle interne consiste à créer la ***liste candidate***, un sous-ensemble de $E(x)$. La liste candidate contient les transformations qui seront évaluées dans le but de choisir celle qui sera appliquée à la solution courante.

Le coût de calcul pour évaluer chaque transformation de la liste candidate dépend de la méthode d'évaluation choisie De plus il ne faut pas oublier que la liste candidate est reconstruite complètement à chaque itération, ce qui implique l'évaluation des éléments de la liste à chaque itération. C'est donc le principal paramètre pour déterminer la taille de la liste candidate. Nous verrons dans la prochaine section des techniques pour créer des listes candidates.

La seconde étape consiste à choisir parmis les transformations de la liste candidate, celle qui est la plus susceptible de faire évoluer la recherche vers la solution optimale d'après la fonction d'évaluation. Le choix de la fonction d'évaluation relève du compromis, un peu comme dans le cas de l'évaluation d'une borne inférieure dans le Branch-and-Bound. On peut choisir une fonction efficace mais dispendieuse en temps de calcul qui force alors à restreindre plus fortement la taille de la liste candidate. Cela peut entraîner l'absence d'exploration d'une sous-région de l'espace de solutions pour un trop grand nombre d'itérations successives. On peut choisir une fonction qui demande peu de temps de calcul mais qui risque d'induire la recherche dans une mauvaise direction. Nous verrons dans une autre section une fonction d'évaluation basée sur des principes de l'intelligence artificielle.

Nous allons faire le choix d'un mode d'évaluation simple (comme c'est le cas de la plupart des applications que nous avons étudiées). Supposons qu'à chaque arête est associé un poids correspondant à la distance entre les noeuds adjacents à l'arête. Soit in_i la somme du poids des arêtes qui entrent dans le parcours pour la transformation i et out_i la somme du poids des arêtes qui sortent du parcours. La transformation e_i qui sera évaluée comme étant la meilleurs dans la liste des candidates, sera telle que $in_i - out_i$ est la plus grande valeur de toutes les transformations appartenant à la liste candidate. Dans le cas où la solution courante correspond à un optimum local, $in_i - out_i$ sera négatif pour toutes les transformations de la liste candidate, on choisit alors la transformation la moins négative. On dira que le mode d'évaluation que nous avons choisi est basé sur la valeur de la fonction objectif. Comme nous le verrons plus loin, cette approche induit parfois la recherche dans une mauvaise direction [23].

Une fois que la meilleure transformation est choisie, la procédure suivante consiste à déterminer, en parcourant la liste tabou, si cette transformation est tabou. Si la trans-

---

sorte de règle d'évaluation des transformations capable de décider de la meilleure transformation selon une perspective à plus long terme. Il ne nous est pas apparu cependant que se soit le but ici du critère d'aspiration.

formation est effectivement tabou, il faut déterminer si l'un des critères d'aspiration peut être appliqué. Dans notre exemple, nous n'avons qu'un critère d'aspiration. Le critère d'aspiration s'appliquera si le résultat de la transformation tabou donne un parcours plus court que tous ceux obtenus depuis le début de la recherche. Si le résultat de ce test est positif, alors le critère d'aspiration s'applique, le statut tabou de la transformation est levé. Par la suite, le critère d'aspiration est remis à jour. Si aucun des critères d'aspiration ne s'applique à une transformation tabou, on enlève cette transformation de la liste candidate et on choisit la meilleure transformation dans ce qui reste de la liste candidate. Si la transformation n'est pas tabou, elle est appliquée et la liste tabou est remise à jour.

Ceci complète la description du fonctionnement de la boucle interne. On peut résumer ce fonctionnement par le pseudo-algorithme suivant:

Procédure 4.1

- 1- Construire une solution initiale;
- 2- Créer une liste candidate;
- 3- Évaluer et ordonner les transformations de la liste candidate;
- 4- Sélectionner la meilleure transformation de la liste candidate;
- 5- Vérifier le statut tabou de la transformation sélectionnée à l'étape 4;
- 6- Si la transformation est non tabou: a) appliquer la transformation à la solution courante, b) ajouter les attributs de la transformation dans les différentes listes tabous, c) retourner à l'étape 2;
- 7- Si la transformation est tabou, vérifier l'applicabilité du ou des critères d'aspiration. Si au moins un des critères est applicable, lever le statut tabou de la transformation et retourner a l'étape 6, sinon retirer cette transformation de la liste candidate et retourner à l'étape 4.

À noter que dans cet algorithme, on présuppose que le processus s'arrête après l'exécution d'un nombre déterminé d'itérations. D'autres critères d'arrêt sont également possibles.

5 Boucle interne: troisième version

Dans les articles les plus récents sur la méthode des tabous on fait usage de l'historique de la recherche, entre autres, pour la formation des listes candidates et l'évaluation des transformations. L'intervention de la mémoire apparaît alors par le biais de procédures inspirées des règles d'apprentissage et de "pattern recognition" en intelligence artificielle. Il nous a semblé que ces adjonctions récentes à la méthode étaient plus significatifs quant au développement futur de la méthode que ce que nous avons vu jusqu'à maintenant. Si l'on considére l'article de Glover de 1986 [14] comme l'un des moments fondateurs de la méthode, ces adjonctions apportent le niveau de flexibilité que Glover pense nécessaire pour résoudre des problèmes d'optimisation combinatoire.

Dans la section qui suit, nous allons examiner ces plus récents développements proposés pour certaines composantes de la mémoire à court terme de la recherche avec tabous.

Tous ces développements ne sont pas redevables d'apport de l'intelligence artificielle mais tous apportent plus de flexibilité au fonctionnement des composantes à court terme. On verra par exemple des propositions, au niveau de la formation des listes candidates, pour contrôler l'explosion combinatoire des solutions. Nous ferons la description de trois modes de gestion dynamique des listes tabous. Nous montrons l'utilisation proposée des règles d'apprentissage de l'intelligence artificielle pour évaluer les transformations.

5.1 Gestion des listes candidates

Bon nombre d'applications de la recherche avec tabous sont des problèmes où le phénomène d'explosion combinatoire se manifeste très clairement. Cette croissance non linéaire du nombre des solutions correspond au très grand nombre de transformations disponibles à chaque itérations. Idéalement on souhaiterait pouvoir évaluer chacune de ces transformations de telle sorte que la liste candidate comprendrait toutes les transformations. Malheureusement cela n'est pas souvent possible à cause du temps de calcul impliqué. C'est la raison pour laquelle Glover [13, 16] suggère un certain nombre de méthodes de constitution de listes candidates s'appliquant sur des sous-ensembles de transformations disponibles. L'objectif de ces méthodes est d'inclure dans les listes candidates des sous-ensembles efficaces d'alternatives pour la recherche dans l'espace solutions [16]. Ces méthodes doivent faire en sorte que la recherche emprunte une trajectoire qui inclut plusieurs bonnes solutions. Nous allons examiner un certain nombre de ces méthodes.

Chacune des méthodes présentées dans cette section implique des modifications ou ajouts aux structures de données de la liste candidate de la section précédente. Pas plus que la section précédente nous n'avons cru bon d'en discuter ici, puisque toutes ces structures sont extrêmement simples.

5.1.1 Méthode par décomposition

Pour plusieurs types de problèmes, les transformations peuvent être divisées naturellement selon les composantes des problèmes. Par exemple, pour un problème de gestion de production, on peut classer les transformations selon les changements qu'elles provoquent pour chaque produit ou pour chaque ressource. Un autre exemple est celui du problème de transport, où l'on peut classifier les transformations selon les changements apporter par région ou par mode de transport. Ceci a inspiré la création de la méthode par *décomposition des transformations* qui construit les listes candidates à partir de sous-ensembles de transformations reliés aux propriétés structurelles du problème [16].

Selon Glover, cette méthode est particulièrement appropriée pour les problèmes de permutations. En effet, si le problème est tel que les transformations consistent à permuter des éléments d'un ensemble (comme dans le TSP), on peut diviser les permutations en sous-séquences successives et inclure dans la liste candidate uniquement les transformations qui impliquent des changements dans une sous-séquence. Cette approche fut utilisée dans une

étude de Laguna et Barnes [44].

Un principe d'apprentissage basé sur l'idée que "good things share common features" [14] fut mis à contribution pour créer des décompositions. Une telle stratégie fut utilisée pour le problème du voyageur de commerce [40]. Dans le but de définir des sous-classes d'arêtes qui sont le plus susceptibles de se retrouver dans de bons parcours, l'algorithme conserve de l'information pendant un certain nombre d'itérations consistant à identifier pour chaque noeud, l'arête adjacente qui apparaît le plus souvent dans les optimum locaux. Ces arêtes sont alors utilisées pour former les listes candidates. Une version très similaire consiste à utiliser après un certain nombre d'itérations, uniquement les arêtes qui se sont retrouvées dans les parcours des optimum locaux.

L'approche par décomposition peut aussi servir de base à un traitement en parallèle de la recherche avec tabous. Chaque décomposition peut être traitée séparément en parallèle, suivie par une phase de fusion des meilleurs solutions de chaque composante et d'une nouvelle décomposition. La répétition du processus se poursuit pour un certain nombre d'itérations.

Notons en terminant, qu'étant donné que la décomposition met l'emphase sur des sous-régions de l'espace de solutions, cette méthode est une forme de stratégie d'intensification de la recherche au niveau de la mémoire à court terme.

5.1.2 Méthode de filtration par niveaux successifs

Cette méthode évalue toutes les transformations disponibles dans un premier temps, en utilisant un critère d'évaluation peu coûteux. Les meilleurs transformations de cette première évaluation sont alors soumises à un deuxième critère d'évaluation plus rigoureux, ce processus pouvant se poursuivre pour plusieurs niveaux.

Le caractère de plus en plus rigoureux des critères d'évaluation à chaque niveau peut se refléter, par exemple, par la prise en compte d'une quantité croissante d'information dans les niveaux successifs (un plus grand nombre de variables ou de contraintes) ou par le choix de méthodes différentes et plus sélectives à chaque niveau successif.

Une autre version de filtration par niveaux successifs consiste à faire des *prospections* (look-ahead) dans les niveaux subséquents en utilisant le même critère d'évaluation pour tous les descendants des meilleurs transformations du premier niveau. Cette approche fonctionne de la façon suivante: soit $E'(x)$ le sous-ensemble des meilleurs transformations de $E(x)$ selon les résultats obtenus au premier niveau et $X(E'(x))$ l'ensemble des solutions qui découlent de ces meilleurs transformations. La fonction $prospection(E'(x))$ consistera à évaluer toutes les transformations de $e(x)$ tel que $x \in X(E'(x))$ et à choisir les k meilleures d'entre elles. Ces prospections peuvent se poursuivre pour plusieurs niveaux successifs. Il est intéressant de noter que le choix des meilleurs transformations à chaque niveau ne dépend pas de leur ancêtre. Il pourrait ainsi arriver que toutes les meilleurs transformations d'un niveau donné découlent du même ancêtre au premier niveau.

Une étude de Fox [9] traite d'une version particulière de prospections appelée "beam

search", où le graphe de recherche des solutions possibles correspond à un arbre (indépendamment du fait que le "beam" lui-même soit un arbre). Dans cette étude le nombre de meilleurs transformations à chaque niveau est constant (k) et le nombre de niveaux peut correspondre aux limites de l'arbre.

Une variante du "beam search", appelée "filtered beam search" [51], utilise deux critères d'évaluation. Le premier critère choisit les c meilleurs transformations de l'itération courante, puis soumet les descendants de ces transformations à un critère beaucoup plus rigoureux. Chacune des c meilleurs transformations donne lieu à un arbre dont chaque noeud est de degré un. Les feuilles de chaque arbre sont utilisées pour choisir les b ($b < c$) transformations de l'itération courante qui formeront la liste candidate.

Comme le souligne Glover, dans le cadre où le graphe de recherche n'a rien d'une structure d'arbre, il est souhaitable que le nombre de niveaux varie au cours de la recherche (et soit limité en profondeur évidemment), et que le nombre de meilleurs transformations à chaque niveau soit fortement restreint. Il nous semble en effet que le coût de calcul de ces prosprections soit un inconvénient à leur application pour bâtir les listes candidates. Nous verrions cependant très bien des variantes de ces méthodes qui seraient utilisées comme sondes exploratoires qui mesureraient "l'épaisseur" ou la distance qui sépare la solution courante d'une région particulièrement intéressante[3].

5.1.3 Méthode de la liste candidate variable

Cette méthode est beaucoup plus élaborée que les deux précédentes. Elle constitue presqu'à elle seule une nouvelle version de la boucle interne de l'algorithme de la méthode des tabous. La méthode de la liste candidate variable est basée sur l'utilisation de deux seuils d'acceptation des transformations. Le *premier seuil* détermine si une transformation est assez bonne pour être admise dans la liste candidate tandis que le *deuxième seuil* détermine en quelque sorte (après un nombre d'itérations) si une transformation est encore assez bonne pour rester dans la liste candidate. À la différence de ce que nous avons vu à la section 5, dans cette méthode la liste candidate n'est pas complètement reconstruite à chaque itération. Le deuxième seuil est le critère qui décide quand la situation s'est détériorée au point qu'il faut recommencer la liste candidate à zéro. Voici d'abord le déroulement des opérations de la boucle interne en tenant compte de l'incorporation de cette méthode.

[3]Ces sondes seraient envoyées dans des directions qui ne semblent pas intéressantes à partir de la solution courante.

Procédure 5.1

- 1a- Sélectioner un nombre de p transformations à examiner comme base pour créer la liste candidate; p est un nombre qui décroit à chaque passage dans cette étape.
- 1b- Parmis les transformations examinées, garder les q meilleures qui formeront la liste candidate. S'il n'y a pas q transformations qui se qualifient, alors continuer à choisir des transformations en dehors des p choisies en 1a, cela jusqu'à ce qu'on obtienne q transformations ou que toutes les transformations disponibles aient été examinées.
- 2- Choisir la meilleure transformation e de la liste candidate.
- 3- Si la transformation e ne satisfait pas le deuxième seuil (toutes les transformations de la liste candidate sont trop mauvaises), aller à l'étape 6.
- 4- Examiner r transformations en dehors de la liste candidate, r est petit par rapport à p. Si une transformation $t_m (m \leq r)$ lors de cette nouvelle recherche a une évaluation supérieure à la transformation t_q trouvée à l'étape 2 alors échanger t_q par t_m et répéter ce processus en continuant le compte pour r à partir de $m+1$. Rafraîchir la liste candidate lors de ce processus en remplaçcant les plus mauvaises transformations dans la liste candidate par celles qui auront été trouvées supérieures lors de l'examen des r transformations.
- 5- Appliquer sur la solution courante la meilleur transformation trouvée à l'étape 4 en tenant compte des conditions imposées par l'algorithme de la recherche avec tabous. Si un certain nombre d'itérations s'est écoulé sans qu'il y ait eu rafraîchissement de la liste candidate, alors aller à l'étape 6, sinon retourner à l'étape 2 pour choisir la prochaine transformation.
- 6- Détruire la liste candidate et retourner à l'étape 1.

Glover [16] n'est pas tellement en faveur de l'utilisation d'un échantillonnage aléatoire simple sur les transformations disponibles pour choisir les p transformations. Il prétend que cela peut créer des poches de solutions non-explorées pour une période trop longue. Il suggère à la place pour l'étape 1, un échantillonnage aléatoire fortement structuré basé sur la version probabiliste de la recherche avec tabous. Selon cette version on affecte des probabilités basées sur l'évaluation relative et le statut tabou de chaque transformation.

Une étude de Frendewey [10], suggère une autre approche pour créer la liste candidate de l'étape 1. Selon cette approche, une transformation est admise dans la liste candidate pourvu que son évaluation soit plus élevée d'un certain pourcentage que celle de la meilleure transformation déjà dans la liste candidate. Expérimentalement on a trouvé que 0.7 était une bonne valeur. Si à la suite de l'ajout d'une très bonne transformation, certaines transformations déjà dans la liste ne satisfont plus ce pourcentage, elles sont rejetées de la liste. Ce processus se continue jusqu'à ce que le nombre de q transformations soit atteint.

Dans la procédure ci-haut, l'étape 4 empêche que la liste candidate repose uniquement sur des éléments qui sont depuis longtemps dans la liste candidate. Cette étape recrée en partie la liste candidate. Plus large est la valeur de r dans l'étape 4 et plus nombreuses peuvent être les itérations dans l'étape 5 avant d'entreprendre de régénérer complètement la liste candidate. Glover fait état d'une étude [13] où r est un triplet (r_1, r_2, r_3) tel que r est d'abord r_1 et si une transformation gagnante est trouvée (c'est-à-dire supérieure à celle choisie à l'étape 2) dans les r_1 premières itérations alors r devient r_2, etc. Finalement r devient r_3 pour le reste du processus, c'est-à-dire que (r_1, r_2, r_3) est équivalent à $(r_1, r_2, r_3, r_3, ...)$. Des valeurs intéressantes dans cette étude pour le triplet

furent (100,50,20) pour des réseaux avec contraintes supplémentaires et de (20,10,10) pour des réseaux avec contraintes normales.

L'étape 5 repose sur un postulat qu'il faut préciser. Lorsqu'une transformation est exécutée sur la solution courante résultant en une nouvelle solution, l'ensemble de transformations disponibles s'en trouve généralement modifié. Selon Glover [16], il y a souvent une association directe entre les transformations disponibles pour des solutions successives qui font qu'on peut traiter ces transformations comme des versions mises à jour de transformations qui existaient auparavant. En d'autres mots, la pertinence de maintenir des éléments dans une liste candidate pour plus d'une itération dépend du fait que l'ensemble des transformations disponibles ne change pas tellement pour une séquence donnée de solutions successives.

Le cas du TSP est un illustration de ce phénomène. Si une transformation correspond à la paire d'arêtes qui est soustraite du parcours, après une mise à jour du parcours (c'est-à-dire l'application de la meilleur transformation sur la solution courante) un grand nombre de transformations demeureront inchangées. Ces transformations inchangées sont celles dont les deux arêtes à soustraire appartiennent à un des deux segments disjoints reconnectés par l'application de la meilleur transformation. La taille de q et de r dépendra dans une certaine mesure du degré de changement apporté au transformations disponibles résultant de l'application de la meilleur transformation sur la solution courante. Dans un tel contexte, pour la constitution de la liste candidate, on a intérêt à utiliser les transformations disponibles qui n'ont pas été modifiées par l'application.

Notons en terminant que les étapes 1 et 4 de la dernière méthode peuvent être implantés en combinaison avec les stratégies de décomposition de la première méthode.

5.2 Gestion dynamique des listes tabous

Les mécanismes simples de gestion de listes tabous qui ont été décris à la section 4 peuvent être modifiés pour permettre une meilleure utilisation des informations fournies par les fonctions d'évaluation. Le but de cette section est de décrire trois façons différentes de gérer dynamiquement les listes tabous.

Nous considérons uniquement la version de ces méthodes qui s'applique pour une seule liste tabou dans chaque application. Ceci nous permettra de ne plus faire de différence entre la base qui rend une transformation tabou et la transformation elle-même. Nous en tirerons un avantage évident pour la limpidité de l'exposé sans toutefois aucune perte de généralité. Toutes ces méthodes ont une extension pour le cas où il y a plusieurs listes tabous.

Les deux premières méthodes font appel à des structures de données et procédures de gestions des listes tabous qui sont fort différentes de celles utilisées pour la gestion statique des listes tabous. Nous discuterons donc dans ces deux cas de l'aspect implantation de ces méthodes de gestion.

5.2.1 Définition du problème

Soit $\{x_1, x_2, ..., x_q\}$ la liste des solutions pour les itérations de 1 à q et soit

$$e_1(x_1), ..., e_{k-1}(x_{k-1}), e_k(x_k), e_{k+1}(x_{k+1}), ..., e_q(x_q)$$

tel que $x_{k+1} = x_{k-1}$. On considére alors que la transformation e_k est l'inverse de la transformation e_{k-1} et on dénote par $\bar{e}_{k-1}$ cette transformation.

La liste tabou *tl* sera telle que

$$tl = \{e_1, e_2, ..., e_i, ..., e_q\}$$

où e_i représente la transformation i appliqué à la solution courante x_i. Contrairement aux sections précédentes, on considérera comme tabou l'inverse $\bar{e}_i$ de la transformation e_i.

Supposons deux solutions x_h et x_k tel que si $h < k$ alors $x_h \neq x_k$ à moins qu'il existe un couplage des éléments $e_h, ..., e_k$ tel que pour chaque paire $\{e_r, e_s\}$ du couplage, $\bar{e}_r = e_s$. Glover [19] nomme cette propriété, ***propriété suffisante***.

Pour les deux premières méthodes de gestion dynamique des listes tabous, nous allons considérer qu'une transformation e_i dans la liste tabou n'implique pas nécessairement que son inverse $\bar{e}_i$ soit tabou, $\bar{e}_i$ est seulement potentiellement tabou du fait de $e_i \in tl$.

5.2.2 Méthode par séquence d'annulations

Dans cette section nous allons supposer que la liste tabou *tl* peut contenir à la fois une transformation e_i et sont inverse $\bar{e}_i$. Si la relaxation de *tl* est telle que $\{e_i, ..., \bar{e}_i, e_q\}$ on dit alors que e_i a été annulée par $\bar{e}_i$.

On défini une nouvelle liste tabou *atl* qui contient uniquement les éléments de *tl* qui n'ont pas été annulés. Par définition, le dernier élément des listes *atl* et *tl* est le même élément e_q.

Ce qu'énonce la "propriété suffisante" c'est que deux solutions x_i et x_l dans la séquence de solutions $\{x_p, ..., x_i, x_j, x_k, x_l, x_m, ..., x_q\}$ ne peuvent pas être la même solution à moins que dans la liste *tl* on ait

$$\{e_p, ..., e_i, e_j, e_k, e_l, ..., e_q\} \tag{1}$$

tel que $e_k = \bar{e}_i$ et $e_l = \bar{e}_j$, c'est-à-dire à moins que chaque transformation dans l'interval correspondant aux solutions $x_i, ..., x_l$ ait été annulée. La liste *tl* suivante

$$\{e_p, ..., e_i, e_j, \bar{e}_i, \bar{e}_j, ..., e_q\} \tag{2}$$

est équivalente à (1). On définira par ***séquence d'annulations*** ou **C-séquence** (pour Cancellation Sequence) la séquence entre la transformation annulée e_j et la transformation qui annule, $\bar{e}_j$, dans (2).

Le fonctionnement de la méthode par séquence d'annulations consiste à spécifier comme tabou une transformation $\bar{e}_i$ si et seulement si la transformation e_i dans la liste *atl* est le dernier élément non annulé d'une C-séquence. Ceci est une condition suffisante pour assurer qu'il n'y aura pas répétition de la même solution dans l'interval couvert par la liste tabou *tl*. Dans l'exemple plus haut, la transformation e_l qui produit la solution x_m à partir de la solution x_l serait tabou du fait que $e_l = \bar{e}_j$ et que e_j est le dernier élément non annulé de la C-séquence $(e_i, e_j, \bar{e}_i)$.

5.2.3 Implantation

Une implantation efficace de cette méthode requiert l'évaluation rapide de chaque élément annulé des C-séquences et l'identification des C-séquences qui ne contiennent qu'un seul élément non annulé. L'implantation proposée par Glover [19] repose sur l'idée que si une C-séquence est comprise dans une autre, la C-séquence la plus longue est dominée par la plus courte et peut donc être oubliée. Dans l'exemple suivant

$$\underbrace{\{e_p, ..., \overbrace{e_j, e_k, e_l, \bar{e}_j}^{a}, ..., \bar{e}_q, e_q\}}_{b}$$

la C-séquence a est comprise dans la C-séquence b. La C-séquence b est dominée par la C-séquence a, en ce sens que s'il y a plus d'une transformation non annulée dans a alors il ne peut pas y avoir une seule transformation non annulée dans b. En retenant un élément dans la C-séquence la plus courte, on tient en fait compte de la plus longue. On appelle *propriété de dominance* cette propriété des C-séquences.

Pour l'implantation, il s'agit de créer deux tableaux, ces tableaux étant définis pour chaque élément actif de *atl*. L'entrée de $st(e)$ contient l'élément f de *atl* qui commence une C-séquence terminée par e. L'entrée de $et(e)$ contient l'élément g de *atl* qui termine une C-séquence commencé par e. Par la propriété de dominance, si e fait partie de deux C-séquences, l'élément f des deux C-séquences qui est le plus près de e est celui tel que $st(e) = f$ (même raisonnement pour g). Donc f et g sont uniquement définis pour chaque e. Lorsque $st(e) = et(e) = e$, on sait que e est le seul élément non annulé dans la C-séquence correspondante.

La procédure fonctionne de la façon suivante: lorsque la meilleure transformation u est choisie par la fonction d'évaluation, on vérifie le statut tabou de u à l'aide de la condition $st(u) = u$. Si u n'est pas tabou, on introduit cette transformation comme dernier élément des listes *atl* et *tl*. On complète la mise à jour de *atl* en vérifiant si u annule une transformation $\bar{u}$. Si c'est le cas, on élimine $\bar{u}$ de la liste *atl*. Finalement on fait la mise à jour des C-séquences en modifiant $st(e)$ et $et(e)$ pour chaque $e \in atl$.

5.2.4 Méthode par élimination inverse

L'idée de base de cette deuxième méthode consiste à tenir compte au niveau des C-séquences de l'addition du nouvel élément dans la liste *atl*. En effet, l'ajout d'un nouvel élément à la liste *atl* protège en quelque sorte contre le retour de certaines solutions; on peut alors tenir compte de ce nouvel élément dans chaque C-séquence.

Pour faire jouer l'influence du nouvel élément dans les C-séquences, la méthode par élimination inverse utilise une approche différente pour détecter qu'un élément est le dernier élément non annulé d'une C-séquence.

L'approche consiste à parcourir en sens inverse (du plus récent élément vers le plus ancien) la liste *tl*. À chaque étape de ce parcours, l'entrée de la liste *tl* est copiée dans une liste temporaire appelée liste de séquences d'annulations résiduelles *lar*. Si un élément $e(i)$ annule un autre élément de la liste *lar*, les deux éléments sont éliminés de la liste *lar*, d'où le qualificatif de résiduelle pour cette liste. Si à tout moment du parcours inverse de *tl*, le nombre d'éléments de la liste *lar* est égal à un, alors l'inverse de l'élément qui se trouve dans *lar* devient tabou.

Ainsi, à chaque itération de la recherche avec tabous, lorsqu'une transformation e_q est ajoutée à *tl*, le parcours inverse de *tl* commence par l'insertion de e_q dans *lar*. $\bar{e}_q$ devient tabou puisque e_q est le seul élément de *lar*. Si à tout moment du parcours inverse, la transformation e_q devient tabou (suite à ce que $\bar{e}_q$ soit la seule transformation dans *lar*), la procédure s'arrête, e_q est enlevé de la liste *tl*, et une nouvelle transformation est choisie pour être appliqué à la solution courante.

On voit que selon cette méthode, le dernier élément e_q ajouté à la liste *tl* comptera pour un élément non annulé dans chaque C-séquence jusqu'à ce que son inverse soit ajouté à la liste *lar* si cet inverse existe dans *tl* ou jusqu'à la fin du parcours inverse de *tl* si cet inverse n'existe pas dans *tl*.

5.2.5 Implantation

L'implantation de cette seconde méthode est la suivante. On utilise deux tableaux, $pred(e)$ et $suc(e)$ qui indiquent respectivement le prédécesseur et le successeur de l'élément e dans la liste *lar*. Si e ne se trouve pas dans la liste résiduelle *lar* à une étape donnée du parcours inverse de *tl*, alors $suc(e) = 0$. On a aussi une variable n qui indique le nombre d'élément dans *lar* à chaque étape du parcours. Le fonctionnement de cette procédure est explicité à partir de la description de la section précédente.

On notera en terminant cette section, que la classification tabou n'est pas la même pour les deux méthodes. Selon Glover, la méthode par élimination inverse produit le minimum d'éléments tabous nécessaires pour éviter la répétition de solutions. La méthode de séquences d'annulations empêche également la reproduction de solutions mais avec plus d'éléments tabous qu'il n'en faut.

5.2.6 Méthode du cycle tabou

La gestion dynamique des listes tabous peut se faire également avec les structures de données et les procédures standards. Nous allons maintenant décrire une dernière version de gestion dynamique de listes tabous où le critère pour déterminer le statut tabou d'une transformation est basé sur la position de cette transformation dans la liste tabou.

Soit $\bar{atl}$ une liste qui contient les inverses des éléments dans la liste atl de la section précédente[4]. L'idée est de partager $\bar{atl}$ en groupes qui seront tabous pour un nombre différent d'itérations. Le statut tabou sera plus fort pour les membres des groupes plus jeunes (les éléments les plus récents de la liste tabou), et plus faible pour les groupes les plus vieux.

La méthode du cycle tabou consiste à rendre tabou chaque groupe k pour t itérations, t étant plus petit pour les vieux groupes. Le nombre d'itérations où un groupe est tabou est appelé *cycle tabou* du groupe k et on le dénote par $tc(k)$. Si un groupe k a un cycle tabou de $tc(k)$ itérations, alors à chaque $tc(k)$ itérations, les éléments du groupe k perdent leur statut tabou et peuvent être choisis comme transformation. On peut représenter ces groupes de la façon suivante:

$$\underbrace{1,2,3,4}_{q},\ldots,\underbrace{f,g}_{k+1},\underbrace{i,j,k}_{k},\underbrace{l,m,n}_{k-1},\ldots,\underbrace{p,q,q+1}_{1}$$

où $q+1$ est la dernière transformation à entrer dans la liste tabou et fait partie du plus jeune groupe, c'est-à-dire le groupe 1.

La procédure fonctionne de la façon suivante. On introduit d'abord un compteur de cycles $cc(k)$ initialisé à 1 pour chaque groupe k, incrémenté de 1 pour chaque itération. Avec $cc(k)$ et $tc(k)$ on définit trois états possibles pour chaque groupe:

fermé	si $cc(k) < tc(k)$
ouvert	si $cc(k) \geq tc(k)$
libre	si le groupe h est **ouvert** $\forall\ h \geq k$.

Un élément du groupe k peut être choisi comme transformation d'une solution seulement si k et tous les autres groupes plus vieux que k sont **libres**.

On assume que $tc(k) < tc(k-1)\ \forall k > 1$. Lorsqu'un élément du groupe k est sélectionné, le compteur $cc(k)$ est mis à jour:

$$cc(k) := cc(k) - tc(k)\ \forall\ h \geq k$$

Selon cette règle, si à une itération i le groupe k devient **libre** et **ouvert** simultanément et qu'une transformation e est choisie dans ce groupe, alors $cc(k) = 0$.

L'objectif de la méthode par cycles tabous, est d'assurer qu'en moyenne, il y aura possibilité de choisir un élément du groupe à chaque $tc(k)$ itérations. D'après Glover [19], la règle de calcul de $cc(k)$ permet d'atteindre cet objectif. Si aucun élément n'a été sélectionné

[4] C'est-à-dire les éléments potentiellement tabou de la méthode par séquence d'annulations.

dans les groupes k à 1 jusqu'à ce que $cc(k)$ soit entre $tc(k)$ et $2tc(k)$, k redeviendra ouvert lorsque le nombre d'itérations atteindra $2tc(k)$.

Le tableau suivant explique le déroulement de la méthode.

	groupe 1		groupe 2		groupe 3	
	cycle tabou = 4		cycle tabou = 3		cycle tabou = 2	
itération	cc(k)	état	cc(k)	état	cc(k)	état
61	1	fermé	1	fermé	1	fermé
62	2	fermé	2	fermé	2	libre*
63	3	fermé	3	ouvert	1	fermé
64	4	libre	4	libre*	2	libre
65	5	ouvert	2	fermé	1	fermé
66	6	libre	3	libre*	2	libre
67	7	ouvert	1	fermé	1	fermé
68	8	ouvert	2	fermé	2	libre
69	9	libre*	3	libre	3	libre
70	6	ouvert	1	fermé	2	libre*

Dans ce tableau les * indiquent qu'une transformation a été choisie dans ce groupe. On voit comment le choix d'une transformation d'un groupe libre affecte le compteur de cycle et les états de chaque groupe. Par exemple, à l'itération 64, le choix d'une transformation du groupe 2 réduit le compteur de cycles des groupes 2 et 3 par la règle suivante[5]:

$$cc(2) := cc(2) - tc(2) = 4 - 3 = 1$$

$$cc(3) := cc(3) - tc(3) = 2 - 2 = 0$$

Il faut mentionner pour terminer qu'il existe une quatrième méthode de gestion des listes tabous qui est une version probabiliste de cette troisième méthode. Dans la quatrième méthode, le choix des transformations de différents groupes est basé essentiellement sur la probabilité $p(k)$ que le groupe k sera libre pour une itération donnée.

[5]Les valeurs de $cc(2)$ et $cc(3)$ à l'itération 65 dans le tableau ne sont pas respectivement de 1 et 0 parce qu'il y a eu aussi l'incrémentation de 1 de ces compteurs au passage de l'itération 64 à 65.

5.3 Évaluation des transformations

Après avoir construit la liste candidate, la recherche avec tabous doit évaluer chacune des transformations dans cette liste et choisir la meilleure évaluation. Jusqu'à maintenant nous supposions un mode d'évaluation relié directement à la valeur de la fonction objectif. Par exemple, dans le problème du voyageur de commerce, la meilleure évaluation était celle qui diminuait le plus ou augmentait le moins la longueur du parcours. Le défaut de cette stratégie est qu'elle n'emprunte pas nécessairement la meilleur trajectoire vers la solution optimale.

D'autre part, une évaluation basée sur la fonction objectif n'est pas toujours applicable puisque la vraie valeur de la fonction objectif n'est pas toujours connue. C'est le cas notamment pour le problème de programmation en nombre entiers lorsqu'on utilise une relaxation de la contrainte d'intégralité comme méthode de résolution [14].

Enfin, il y a des cas où on ne pense calculer la vraie solution qui résulterait de chaque transformation de la liste candidate. Par exemple, si une procédure de résolution fait appel à un algorithme du simplexe à chaque iteration, il serait très coûteux d'appliquer le simplexe pour chaque élément de la liste candidate en vue de connaître la meilleur solution.

Ce sont là plusieurs raisons pour développer des outils mieux adapté à l'évaluation des transformations dans la liste candidate. L'*analyse par objectifs* (target analysis) est l'un de ces outils qui fut développé par Glover et Greenberg [14, 24].

Cette section est consacrée à une introduction très brève de l'analyse par objectifs. Nous reviendrons sur cette méthode dans la section consacrée aux applications de la recherche avec tabous. Nous montrerons alors un exemple d'utilisation de l'analyse par objectifs dans une application de la méthode des tabous au problème d'ordonnancement.

5.3.1 Analyse par objectifs

Nous dirons qu'une transformation e à l'itération i est *améliorante* si le fait d'appliquer e à x_i est tel que $c(x_{i+1}) < c(x_i)$ à l'itération $i+1$. Si $c(x_{i+1}) \geq c(x_i)$ à l'itération $i+1$, on dira que e est une transformation *non améliorante*.

L'analyse par objectifs avait été développé dans le but d'utiliser les techniques de "pattern recognition dans le temps" [14] pour donner aux heuristiques l'habilité d'apprendre quelles sont les meilleurs règles pour résoudre chaque classe particulière de problèmes.

> The ultimate goal of target analysis is to create some form of "master decision rule" which will assist a heuristic or optimal method in making better choices (or moves) during the solution procedure [23].

L'analyse par objectifs consiste à investir des efforts pour résoudre, à l'aide d'une méthode connue, une série de problèmes tests représentatifs d'une certaine classe de problèmes. Les solutions optimales ou presque optimales qui résultent de ces tests sont appelées *solutions cibles*. Par la suite, les mêmes problèmes sont de nouveau résolus en vue cette fois d'évaluer chaque transformation en fonction de sa capacité à diriger la trajectoire de

recherche vers la solution cible. Cette évaluation s'exécute en attribuant à chaque transformation un certain pointage. Ces pointages servent par la suite à définir de nouveaux évaluateurs capable d'identifier les transformations qui auraient eu de bons pointages.

On peut détailler le schéma précédent sous la forme d'une procédure en 5 étapes:

Procédure 5.2

- 1- Utiliser une heuristique connue pour trouver la solution optimale ou presque optimale aux problèmes tests;
- 2- Résoudre de nouveau chaque problème en qualifiant de bonne ou mauvaise chaque transformation en fonction de son habilité à faire progresser la recherche vers la solution cible;
- 3- Définir les règles qui conduisent à choisir les transformations qui ont été qualifiées comme bonnes dans un cadre où on ne dispose par des solutions cibles;
- 4- Créer une règle de décision qui incorpore les différentes règles trouvées dans l'étape précédente. Cette règle de décision sera l'évaluateur des transformations. Définir un modèle qui servira à trouver les valeurs des différents paramètres de la règle de décision;
- 5- Appliquer la règle de décision aux problèmes tests pour évaluer son efficacité.

Lorsqu'une fonction d'évaluation est basée sur la fonction objectif, les transformations sont classées en deux catégories: améliorantes et non améliorantes. La conclusion de l'étude de Glover et Laguna [23] pour le problème d'ordonnancement, est que lorsqu'il y a plusieurs transformations améliorantes, une fonction d'évaluation uniquement basée sur la fonction objectif est efficace dans sa capacité à guider la recherche vers la solution optimale. Cependant, lorsqu'il n'y a aucune transformation améliorante, ce mode d'évaluation performe médiocrement. Dans ce cas, l'analyse par objectifs a permis de construire une fonction d'évaluation mixe basée sur la fonction objectif lorsqu'il y a des transformations améliorantes et sur d'autres critères dans le cas contraire. Ceci a permis d'améliorer sensiblement l'efficacité de la recherche avec tabous.

Glover donne de nombreux exemples où on peut trouver une possibilité d'application de l'analyse par objectif au niveau de la méthode des tabous [18, 20, 23] ou d'autres méthodes [25]. En fait, l'approche de l'analyse par objectifs peut être utilisée partout où il y a des règles de décisions. Elle peut contribuer à définir les valeurs des paramètres de ces règles, rendre plus précise ces règles, trouver d'autres règles mieux adaptées.

6 Composantes à moyen et long terme

La recherche basée uniquement sur les composantes de la mémoire à court terme de la méthode des tabous est loin d'être inefficace. Selon Glover [15], bon nombre des meilleurs solutions trouvées par la méthode des tabous apparaîssent dès la première phase de la méthode, et n'impliquent donc pas l'utilisation de fonctions basées sur les autres niveaux de la mémoire. Cependant l'utilisation des fonctions de la mémoire à moyen et long terme devient nécessaire pour bon nombre de problèmes complexes et pour les exemplaires de

grande taille des problèmes plus simples. Ces fonctions aident à trouver de meilleurs solutions soit en intensifiant la recherche sur une région particulière, soit en diversifiant la recherche sur l'espace global de solutions.

6.1 Intensification

La phase d'intensification de la méthode suit en général une phase d'exploration basée sur les fonctions de la mémoire à court terme. La détermination des zones d'intensification est fonction des solutions trouvées par la mémoire à court terme. Les méthodes d'intensification elles-mêmes reposent largement sur la nature combinatoire des problèmes traités par la recherche avec tabous.

On sait que la solution d'un problème combinatoire consiste souvent à permuter des éléments ou à combiner d'un sous-ensemble des éléments qui constituent l'input du programme (les arêtes, les noeuds, les variables, contraintes, etc.). Cette propriété des problèmes combinatoire permet l'implantation d'un mode d'intensification basé sur un tableau des fréquences d'apparition de chaque élément dans une solution de la mémoire à court terme. La phase d'intensification consiste à construire des solutions par la permutation des éléments qui ont les plus grandes fréquences. Ces permutations définissent une ou des régions de l'espace de solutions qui se caractérisent par de bonnes solutions. On espère qu'en permutant ces éléments on réussira à trouver la meilleur solution dans cette région si on ne l'avait pas déjà trouvée dans la phase d'exploration de la mémoire à court terme.

Une autre stratégie pour faire de l'intensification consiste à limiter l'espace global de solutions aux seules éléments qui sont apparus dans des solutions trouvées lors de la phase d'exploration de la mémoire à court terme. Pour le TSP par exemple, le nombre d'arêtes impliqué dans une phase d'exploration de la boucle interne est généralement une fraction de l'ensemble des arêtes. De plus, comme on l'a déjà vu, les éléments dans les solutions consécutives ont tendance à ne pas varier tellement (il n'y a que deux arêtes qui changent à chaque solution). En conséquence, en utilisant uniquement ce sous-ensemble des arêtes, on se trouve à résoudre un problème de taille beaucoup moindre. L'exploration se fait dans une on quelques régions voisines et la recherche progresse très rapidement à cause de la taille relativement petite de ce sous-problème.

6.2 Diversification

En relation avec la fonction $c(x)$ du problème de minimisation P, nous allons considérer que la fonction $c(x)$ peut ne pas être linéaire et dans certaines stratégies $x \notin X$ peut apparaître (parcours de l'espace non réalisable pour un certain nombre d'itérations).

Nous allons examiner deux opportunités de diversification. L'une est basée sur la notion de distance d'une transformation, l'autre sur le redémarrage de la recherche dans une nouvelle région de l'espace de solutions.

6.2.1 Diversification basée sur la distance des transformations

Selon Glover, il existe des situations où les meilleurs solutions ne peuvent être obtenues qu'en traversant des régions très sous-optimales ou des espaces de solutions non réalisables. Dans ces situations, il pense que l'utilisation de stratégies de diversification basées sur la distance des transformations sont appropriées. Cette approche fut implantée pour un problème de partition de graphe [21].

L'idée qui soutend cette approche est que les transformations susceptibles de trouver deux solutions consécutives éloignées ont peu de chance d'être intéressantes pour une fonction d'évaluation normale. Glover exprime de la façon suivante cette notion de distance d'une transformation:

> The notion of move distance derives from the fact that some moves create greater changes in the current solution than others. For example, in the scheduling application (voir le premier problème dans la section des applications), a move that transfers a task to a new sequence location several positions away from its current position involves a greater move distance than one that transfers a task to an adjacent location. Similarly, in an integer programming context, the degree to which a move changes the relative feasibility of certain constraints or alters the value of certain dependent variables can be made the basis for defining a distance measure. [18]

Toujours selon Glover, la relation avec la fonction d'évaluation dérive de la corrélation suivante:

> ...moves that involves greater distances are likely to entail greater cost, as determined by a standard evaluator...this implies that large distance move are likely to appear relatively unattractive by such evaluators...(even though they) may be necessary to cross humps to better solutions. [18]

Dans le but d'implanter une telle forme de diversification, il est bon de regrouper les transformations selon certain intervals de distances. À partir d'informations basées sur l'historique de la recherche à court terme on peut déterminer les transformations dans chaque classe qui semblent les plus intéressantes. Cette évaluation peut se faire à l'aide de méthodes basées sur l'analyse par objectifs [18]. Lorsque la ou les fonctions d'évaluation standard semblent perdre de leur efficacité, le contrôle de l'algorithme des tabous est retourné à la boucle externe qui applique alors les meilleurs transformations des classes de distances les moins solicitées durant la phase d'exploration à court terme.

6.2.2 Diversification par solutions de redémarrage

Cette deuxième stratégie de diversification est commune à beaucoup d'heuristiques de recherche. Ce qui différencie la recherche avec tabous c'est que la solution de redémarrage

est induite par l'historique de la recherche tandis que la plupart des autres heuristiques utilisent une solution générée aléatoirement. L'objectif de la recherche avec tabous est de générer des points de redémarrage qui soient aussi différents que possible des solutions obtenues auparavant.

L'implantation de cette stratégie de diversification est simple. On utilise le même tableau que celui des stratégies d'intensification. Une solution de redémarrage est formée d'un sous-ensemble des éléments du tableau ayant les plus petites fréquences et constituant une solution réalisable.

7 Version probabiliste et oscillation stratégique

Il y a deux brèves sections de la recherche avec tabous qu'il ne nous a pas été possible de couvrir à cause du manque de temps et de l'impossibilité d'obtenir certains documents. Ces sections sont la version probabiliste de la recherche avec tabous et la notion d'oscillation stratégique. Nous faisons ici une très brève introduction à ces deux concepts.

7.1 Oscillation stratégique

Lorsqu'on exécute une recherche avec la méthode des tabous, on observe que la séquence des valeurs de la fonction objectif n'est plus monotone. La valeur de cette fonction oscille. Cette variation est causée par l'influence des listes tabous. Glover croit qu'il y a d'autres paramètres que la fonction objectif qu'il peut être utile de manipuler de cette façon. C'est le cas pour des problèmes ayant des structures particulières comme le problème du sac à dos et les problèmes de partitionnement et le recouvrement d'ensembles. L'oscillation stratégique consiste à définir de nouvelles listes tabous pour influencer certains paramètres et amener la recherche à parcourir de nouvelles régions. Glover pense que l'oscillation stratégique est particulièrement utile lorsque les paramètres naturels d'une application contraignent à d'étroites bandes les régions des solutions réalisables. L'oscillation stratégique peut être pratique pour s'éloigner des frontières de ces régions ou pour traverser des espaces de solutions non réalisables dans le but d'accélérer la recherche [18, 24].

7.2 Recherche probabiliste avec tabous

La version probabiliste de la recherche avec tabous est une variante de la version déterministe où l'on remplace les règles décrites depuis le début de ce travail par des probabilités. De cette façon, on peut par exemple remplacer l'évaluation des transformations par une probabilité qui reflète son évaluation [17, 18]. Nous avons fait mention déjà dans la section 5.2 d'une méthode probabiliste de gestion des listes tabous. L'article de Faigle et Kern [6] est une référence que nous aurions aimé consulter concernant cette version probabiliste de la recherche avec tabous.

8 Applications

Bien que la recherche avec tabous soit une méthode dont le développement est récent, il y a déjà plusieurs problèmes où cette méthode fut appliquée avec succès. Nous avons recensé dans la littérature les applications suivantes dans différents domaines d'optimisation combinatoire.

Hertz et De Werra [29, 30], Friden, Hertz et De Werra [11, 12], Wendelin [57] ont appliqué la méthode des tabous à des problèmes de graphe ou de flots dans les réseaux. Laguna, Barnes et Glover [23, 44], Eck [5], Laguna [43], Glover et McMillan [22], Bovet et De Werra [1], Winder et Hertz [58] ont appliqué la méthode des tabous à des problèmes d'ordonnancement, d'affectation ou de confection d'horaires. Le problème du voyageur de commerce a été l'objet de plusieurs applications: Knox et Glover [41], Malek, Heap, Kapur et Mourad [47], Malek, Guruswamy, Owens et Pandya [46]. Finalement, la méthode des tabous a aussi été appliqué à un problème de télécommunication, Oliveira et Stroud [50], de logique probabiliste, Jaumard, Hansen et Aragao [34], de reconnaissance dans les réseaux de neurones, De Werra et Hertz [4], de design de topologies de réseaux, Lee [45].

Dans le cadre d'un travail comme celui-ci, il nous était évidemment impossible de consulter chaque étude et d'en faire un résumé. Nous nous sommes donc limité à une sélection de trois des applications citées plus haut. Nos critères de sélections ont été fonction de nos capacités à comprendre rapidement ces applications, nos intérêts particuliers et notre évaluation des forces qui risquent d'influencer le développement futur de cette méthode. Nous allons décrire une application en parallèle de la recherche avec tabous parce qu'effectivement le parallélisme risque d'avoir un impact sur cette méthode. L'utilisation de la recherche avec tabous en conjonction avec d'autres méthodes est un thème cher à Glover. L'application de la méthode des tabous aux réseaux de neurones et l'application de l'analyse par objectifs à la méthode des tabous sont des exemples de l'influence mutuelle que peuvent avoir ces conjonctions.

8.1 Analyse par objectifs et recherche avec tabous

Cette application a pour but de faire ressortir une utilisation de l'analyse par objectifs dans le cadre de la recherche avec tabous. Cette application fait référence en entier à un rapport technique de Glover et Laguna sur l'analyse par objectifs et la recherche avec tabous [23].

8.2 Définition du problème

Nous considérons le problème d'ordonnancement suivant: N tâches doivent être ordonnancées sur une machine dans le but de minimiser la somme des coûts qui sont composés du "setup" de chaque tâche et de pénalités associées aux délais. Chaque tâche ($i = 1, 2, ..., n$) prend t_i unités de temps sur la machine, il y a une pénalité de p_i pour chaque unité de temps que la tâche est retardée après le temps $t = 0$. $s_{i,j}$ est le coût

du "setup" de la tâche j après la tâche i. Deux tâches bidon 0 et $n+1$ sont créés dans chaque ordonnancement, $t_0 = t_{n+1} = 0$ et $p_0 = p_{n+1} = 0$. Un ordonnancement à la forme $\prod = \{0, \pi(1), \pi(2), ..., \pi(n), n+1\}$ où $\pi(i)$ est l'index de la tâche en position i de l'ordonnancement. Toute permutation des n tâches est un ordonnancement réalisable. Le modèle est le suivant:

Minimiser $F(\prod) = D(\prod) + S(\prod)$

où $D(\prod) = \sum_{i=1}^{n} d_{\pi(i)} p_{\pi(i)}$

$S(\prod) = s_{0,\pi(1)} + \sum_{i=1}^{n-1} s_{\pi(i),\pi(i+1)} + s_{\pi(n),n+1}$

et $d_{\pi(i)} = \sum_{j=1}^{i-1} t_{\pi(i)}, i = 2, 3, ..., n$ et $d_{\pi(1)} = 0.$

8.3 Description de l'application de la méthode des tabous

La procédure de la recherche avec tabous est la suivante:

Procédure 8.1

Initialisation
do
  Constituer la liste candidate;
  Evaluer les transformations;
  Choisir la meilleur transformation;
  Appliquer la transformation;
  $F(\prod) = F(\prod)$ + valeur de la meilleur transformation;
  If $F(\prod) < F(\prod^*)$
    $\prod^* := \prod$;
    $F(\prod^*) := F(\prod)$
  end if
end do

On considére deux classes de transformations dans cette implantation de la méthode des tabous. La première consiste en un échange de deux tâches (swap), la deuxième est un transfer, c'est-à-dire qu'on prend une tâche de sa position actuelle et on l'insère dans une autre position de l'ordonnancement. À chaque itération, il y a $n(n-1)/2$ swaps disponibles et $n(n-3)+2$ transfers disponibles pour constituer la liste candidate. On dénote par $T(\pi(i), j)$ le transfer de la tâche $\pi(i)$ entre les tâches $\pi(j-1)$ et $\pi(j)$ et par $S(\pi(i, \pi(j))$ le swap des tâches entre les positions i et $j, i < j$. Dans le cas du transfer comme dans celui du swap, $\pi(i)$ est l'attribut des transformations qui sera classifié tabou. À chaque

itération, la transformation qui est choisie pour être appliquée à la solution courante est celle qui est la meilleur parmi les deux classes. L'évaluation de chaque transformation est faite de la façon suivante: $F(\prod'') - F(\prod')$ où $\prod'$ et $\prod''$ sont les solutions avant et après la transformation. Le critère d'aspiration sera le suivant: une transformation perdra son statut tabou si son application produit la meilleur solution depuis le début de la recherche.

8.3.1 Le modèle d'analyse par objectifs

Le lecteur voudra se référer à la procédure 5.2 pour cette sous-section.

Dans cette étude, 4 problèmes furent choisis comme problèmes tests. La taille des problèmes était relativement restreinte, on connaissait leur solution optimale quoique par la recherche avec tabous on n'avait pas réussi à trouver cette solution optimale en un nombre spécifié d'itérations.

Pour la deuxième étape de la procédure 5.2, on a défini deux pointages comme mesure d'évaluation du succès de chaque transformation à rapprocher la recherche des solutions cibles. Un premier pointage est le *positionnement absolu*, un paramètre qui compte le nombre de tâches qui occupent la même position que dans la solution optimale. Le deuxième pointage est le *positionnement relatif* qui compte le nombre de tâches qui sont placées immédiatement après la position de leur prédécesseur dans la solution optimale. Pour la solution optimale, la valeur du positionnement absolu est de n et celle du positionnement relatif est de $n + 1$ pour une valeur cible de ces pointages de $2n + 1$. Le pointage attribuable à chaque transformation est le pointage de l'ordonnancement (solution) après la transformation moins le pointage de l'ordonnancement avant la transformation.

Dans cette étude, l'évaluation d'une transformation à l'aide de la fonction objectif consiste à choisir comme meilleur transformation celle qui améliore le plus la fonction objectif ou encore qui la détériore le moins. On a donc deux concepts d'évaluation, l'un relié à la fonction objectif, l'autre étant celui des pointages.

Sous ce schème, on a résolu de nouveau les problèmes à l'aide de la recherche avec tabous mais en classifiant les meilleurs transformations à chaque itération en deux classes: les transformations améliorantes et les transformations non-améliorantes. La valeur des pointages des positionnements absolu et relatifs était enregistré séparément selon que la transformation était améliorante ou pas. Les résultats de cette expérience ont montré que les transformations non-améliorantes avaient tendance à placer les tâches en dehors de leur position cible tandis que le contraire se produisait pour les transformations améliorantes. L'interprétation de ces résultats est que l'évaluation à l'aide de la fonction objectif performe bien lorsque la transformation est améliorante mais qu'une stratégie différente devrait être utilisée pour choisir les transformations pour les itérations où aucune transformation améliorante n'est disponible.

Lorsque les pointages sont connus pour chaque transformation de chaque problème test, on utilise ces pointages comme cibles pour la création de nouvelles fonctions d'évaluation. C'est-à-dire, dans le contexte des problèmes autres que les problèmes tests, on essaie de

créer des fonctions qui vont aussi souvent que possible prendre des décisions correspondant à l'évaluation par pointage. L'étape précédente a permis d'identifier les trois fonctions d'évaluation suivantes:

- 1- Fonction d'évaluation basée sur la valeur de la fonction objectif.
- 2- Fonction d'évaluation basée sur le positionnement absolu. Cette fonction garde la trace des positions absolues occupées par chaque tâche de haute qualité au niveau de la solution (c'est-à-dire des solutions dont la fonction objectif ne s'éloignait pas plus que d'un certain pourcentage de la meilleur fonction objectif trouvée jusqu'à l'itération courante). Cette trace est gardée dans une matrice de positionnements P de tâches versus les positions. La valeur d'une entrée est incrémentée à chaque fois qu'une tâche apparaît dans une position pour une solution de haute qualité.
- 3- Fonction d'évaluation basée sur le positionnement relatif. Même que pour la fonction précédente appliqué aux positionnements relatifs.

On comprend que les solutions qui se retrouvent dans les matrices P et P' jouent le rôle de solutions cibles de la méthode des pointages. La transformation qui recevrait la meilleur évaluation selon la fonction deux par exemple, serait celle qui produirait la solution avec le plus haut pointage selon les entrées de la matrice P.

L'étape 4 de la procédure 5.2 consiste à formuler une règle de décision basée sur l'incorporation des trois fonctions d'évaluation. Ceci conduit à définir un modèle d'expérimentation qui sera utilisé pour fixer les valeurs des différents paramètres de la règle de décision.

Ce modèle est basée sur une variante de l'enregistrement des positions absolues (matrice P) et de même qu'une variante de l'enregistrement des positions relatives (matrice P'). Il s'agit de normaliser les entrées de chaque ligne des matrices de positionnnement en divisant les entrées de chaque ligne de ces matrices par la somme des entrées de la même ligne. On normalise les colonnes également de la même façon. La normalisation de l'enregistrement des positions absolues par exemple, peut être interprétée comme étant la création de probabilités pour l'affectation de la tâche aux diverses positions tandis que la normalisation des colonnes sera considérée comme la probabilité d'affecter une position donnée à chaque tâche. Une interprétation similaire peut être donnée à la matrice des positions relatives. Chaque entrée de la ligne normalisée peut être considérée comme une position fantôme pour la tâche associée dans le sens où la tâche n'est généralement pas ni complètement présente ni complètement absente de cette position mais plutôt occupe cette position avec une certaine probabilité. De façon similaire, pour les entrées de la colonne normalisée où chaque entrée décrit une tâche fantôme où la position occupée contient avec une certaine probabilité. Vue de cette façon, la matrice normalisée peut-être vue comme des ordonnancements fantômes avec un taux d'occupation probabiliste. L'évaluation des transformations par les matrices normalisées se fait de la même manière que pour les fonctions 2 et 3 plus haut [6].

[6] La raison de l'utilisation des matrices normalisées est de corriger un certain biais des matrices non

Il est possible de concevoir un système d'évaluation moins coûteux en temps de calcul en donnant une approximation de celui des matrices par un vraie ordonnancement cible. Le choix logique pour un tel ordonnancement cible est le meilleur ordonnancement obtenue jusqu'à l'itération présente. La règle de décision qui en résulterait serait d'appliquer l'ordonnancement cible uniquement pour les transformations non-améliorantes, utilisant la fonction d'évaluation basée sur la fonction objectif pour les transformations améliorantes.

Les tests effectués en utilisant cette règle de décision on permis à la recherche avec tabous de trouver la solution optimale pour tous les problèmes tests en moins de 175 itérations pour chacun, tandis que sans cette règle le nombre d'itérations dépassait 250.

8.4 Version parallèle de l'algorithme des tabous

Nous allons maintenant traiter d'une version parallèle de la recherche avec tabous. Dans la consultation que nous avons fait des articles consacrées à la méthode des tabous, nous avons retrouvé deux articles faisant référence à une implantation en parallèle de cette méthode. Malheureusement, il ne nous a pas été possible d'avoir accès à un de ces articles [50].

Glover, dans quelques uns de ces articles, suggère comment nous pourrions utiliser le parallélisme pour appliquer sa méthode [15, 16, 18]. Nous avons vu dans la section 5.1.1 sur les listes candidates que la méthode par décomposition offrait une base pour exécuter le tabou en parallèle. Glover souligne également la diversification par points de redémarrage comme moyen de faire du parallélisme en tabou. Lors de la diversification, un certain nombre de points de redémarrage serait générés qui constitueraient la solution initiale de chaque processus en parallèle. Glover suggère que le parallélisme soit un moyen de rendre plus efficace la diversification basée sur la notion de distance des transformations (voir 6.2.1). L'évaluation de ce type de transformation est plus difficile et Glover pense que l'on peut utiliser un processus en parallèle pour évaluer les "conséquences" de celles de ces transformations qui sont considérées comme les meilleurs. Il est également question d'un mode de "décomposition par niveaux" [14] de la recherche avec tabous qui donnerait prise à un traitement en parallèle. Nous n'avons pas eu le temps malheureusement d'explorer longuement dans cette direction.

Il est intéressant de noter que Glover semble estimer que la mémoire à long terme est le meilleur lieu d'application du parallélisme. Dans l'exemple que nous allons étudier de plus près, nous verrons que le parallélisme se substitue en quelque sorte à cette mémoire à long terme. Au début de la section 6 nous avons vu que pour un problème complexe ou de grande taille, il est utile d'utiliser les composantes à moyen et long terme de la méthode. Nous pensons qu'il ne serait pas inutile de se demander jusqu'à quel point il ne serait pas plus naturel de concevoir la mémoire à long terme uniquement comme un processus parallèle. Quelques unes des méthodes dans l'introduction de ce travail s'implantent plus naturellement en parallèle, il nous semble que se soit peut-être le cas pour ces composantes à long terme de la recherche avec tabous.

---

normalisées à favoriser les solutions instables (c'est-à-dire celles qui apparaissent dans plus d'une position des matrices).

Cette section consacré à une version parallèle de la méthode des tabous réfère entièrement à un article de Malek et al. de 1989 [46]. Il s'agit d'une application de la recherche avec tabous au problème du voyageur de commerce. Un des objectifs de cette étude consistait à vérifier si la solution obtenue sur le problème du TSP serait meilleur avec une implantation parallèle de la méthode des tabous. Plus spécifiquement, pour un temps de calcul à peu près équivalent on a comparé la longueur des parcours obtenus, et pour un même parcours on a comparé les temps de calcul.

8.4.1 Définition de l'heuristique des tabous

Dans cette étude, un parcours est un vecteur dont chaque indice i représente une ville et chaque entrée du vecteur indique la position de la ville d'indice i dans le parcours. Comme dans l'exemple d'application au TSP de la section 4, une transformation est un swap 2-opt, le critère d'aspiration est la meilleur solution obtenue avant l'itération courante. Soit i le contenu de l'entrée d'indice v avant l'application de la meilleur transformation à la solution de l'itération k et soit j le contenu de l'entrée v à l'itération $k+1$. Si $j > i$, on dira que la ville v s'est déplacée à droite dans le cycle, et à gauche si $j < i$. Pour cette expérience, la liste candidate comprend tous les swaps disponibles à chaque itération. Dans le cas où il existe des transformations améliorantes, le critère d'évaluation des transformations est l'amélioration de la fonction objectif, la moindre détérioration étant le critère dans le cas où il n'y a que des transformations non-améliorantes. La fonction d'évaluation dépend donc uniquement de la fonction objectif. Le critère d'arrêt est un nombre prédéterminé d'itérations.

Les auteurs de la recherche ont trouvés 9 attributs possibles comme base pour déterminer le statut tabou des transformations. Pour chaque test cependant, ils n'utilisent qu'un seul attribut à la fois. Ces 9 attributs sont reliés soit aux villes impliquées dans les swaps, soit à la position des villes avant et après le swap, soit à la direction de leur déplacement. Les 9 attributs sont les suivants:

- 1- **Vecteur(i,j,position(i),position(j))**. Cet attribut empêche que les villes i et j soient dans un cycle aux position i et j respectivement après l'exécution d'une transformation.
- 2- **Vecteur(i,j,position(i),position(j))**. Le même vecteur mais cette fois pour empêcher que les villes i et j n'occupent les positions i et j respectivement.
- 3- **Vecteur(i,position(i))** pour empêcher que la ville i retourne à la position i.
- 4- **Ville i** pour empêcher que la ville i se déplace à gauche de sa position courante.
- 5- **Ville i** pour empêcher la ville i de se déplacer.
- 6- **Vecteur(j,position(j))** pour empêcher que la ville j ne retourne à la position j.
- 7- **Ville j** pour empêcher que la ville j ne se déplace à droite de sa position courante.
- 8- **Ville j** pour empêcher la ville j de se déplacer.
- 9- **Ville i et j** pour empêcher les deux villes de se déplacer.

Les attributs 3 jusqu'à 9 ont été établis en assumant que chaque ville i et j est telle que position(i) < position(j). La condition 1 est la moins contraignante quand au nombre de transformations disponibles à chaque itération, la condition 9 étant la plus contraignante. L'ordre des conditions indique une croissance dans le degré de restriction.

Les auteurs font remarquer que le choix de l'attribut et la taille de la liste tabou sont deux paramètres très interdépendants. Ils ont remarqué que les attributs plus contraignants permettent une taille optimale de la liste tabou qui est plus courte que celle des tests avec des attributs moins contraignants. D'après leurs observations, les attributs 1, 2, 3 et 6 nécessitent des listes tabous de taille à peu près équivalente à celle de la taille du problème (c'est-à-dire au nombre de villes qui sont dans le parcours). Pour les attributs 4 et 7, une liste tabou de taille de 1/4 à 1/3 de celle du problème était efficace. Pour les attributs 5, 8 et 9, 1/5 de la taille du problème permettait d'empêcher le cyclage tout en ne passant pas à coté d'une bonne solution.

Pour les fonctions de la mémoire à court terme, il n'y a pas de différence entre les versions séquentielle et parallèle de la méthode des tabous. L'implantation séquentielle est cependant la seule à faire appel à des fonctions de la mémoire à long terme. La mémoire à long terme est utilisée pour faire de l'exploration de nouvelles régions (c'est-à-dire essentiellement de la diversification). Elle est basée sur la fréquence d'apparition des arêtes du graphe dans le parcours. Après un certain nombre d'itérations, cette fonction crée un nouveau parcours à partir d'un certain nombre des arêtes qui possèdent les fréquences les plus faibles. Nous verrons plus loin pourquoi il n'existe pas de fonction basée sur la mémoire à long terme dans la version parallèle.

8.4.2 Machine utilisée

Cette expérimentation fut réalise sur le Sequent 8000, un système à mémoire partagée avec 10 processeurs. Le système d'exploitation de ce système est Dynix, une version de Unix pour l'exploitation de systèmes parallèles. Le langage utilisé est PPL, une extension de C contenant des primitives pour la programmation en parallèle. Nous avons eu déjà l'occasion d'utiliser ce langage et un Sequent de 8 processeurs.

8.4.3 Implantation en parallèle

Les algorithmes parallèles se divisent en deux grandes catégories: les algorithmes qui exécutent plusieurs fonctions différentes sur les mêmes données et ceux qui exécutent la même fonction sur des données différentes. Nous pensons que cet algorithme parallèle de la méthode des tabous se range dans la deuxième catégorie. L'approche que les auteurs utilisent pour l'implantation en parallèle de la méthode des tabous consiste à lancer l'algorithme séquentiel du tabou sur plusieurs processeurs en même temps, tout en divisant l'espace de recherche à l'aide de valeurs différentes pour les paramètres d'entrés de chaque processus. Ceci nous semble être une procédure assez commune dans le domaine

des algorithmes parallèles de recherche [52, 54].

Le fonctionnement de la recherche avec tabous en parallèle est le suivant. Au départ, selon une configuration maître-esclaves, le processus maître copie l'algorithme du tabou sur chacun des processeurs esclaves, puis sélectionne et attribue à chaque processus les valeurs des paramètres en entrée. Ces valeurs sont choisies de telle sorte que chaque processus explorera son propre espace de recherche. Après un certain nombre d'itérations, les processus esclaves s'interrompent et après comparaison, le processus maître copie dans chaque processeur la configuration du processus qui avait la meilleur solution. Un ensemble de valeurs pour les paramètres est alors une fois encore fourni en entrée à chaque processus, les listes tabous sont mises à vide et le calcul recommence. Ce processus pour la recherche avec tabous en parallèle se poursuit jusqu'à ce qu'un nombre prédéfini d'itérations se soient exécutées.

Les algorithmes séquentiels s'exécutant sur chaque processeurs ne sont pas différents de l'algorithme séquentiel normal du tabou, sauf au niveau de la fonction de diversification de la mémoire à long terme. Cette implantation parallèle n'effectue pas de diversification basée sur la mémoire à long terme. La raison en est que pour tirer profit de l'exploitation en parallèle, il faut que les comparaisons par le processus maître des résultats des processus esclaves ne soient pas trop distantes. Le fait d'intercaller entre deux comparaisons l'exécution d'un processus de redémarrage de la recherche rendrait nul pour la recherche le résultat de la comparaison par le processus maître et chaque comparaison annulerait l'effet d'un repositionnement par la mémoire à long terme. Les auteurs soulignent qu'il serait peut être utile de faire de la diversification pour les problèmes de très grande taille, là où le nombre d'itérations entre chaque comparaison le justifierait.

8.4.4 Résultats de l'expérimentation

Chaque taille de problème et chaque ensemble de paramètres fut testé en séquentiel (nous supposons avec un seul processeur du Sequent), puis en parallèle sur 4 processeurs[7]. Le choix de la valeur des paramètres devait faire en sorte que chaque processus explore une région différente de l'espace de recherche.

Pour des tailles de problèmes de 25, 33 et 42 villes, il y a eu peu de différences entre les explorations en séquentielles et celles en parallèles. La raison est que le temps d'exécution de la version séquentielle est déjà très court et cette version trouve d'excellents parcours. Pour une taille de 57, la version parallèle trouvait la solution optimale plus souvent et plus rapidement qu'en séquentiel. Avec un problème de taille 75, la solution obtenue en parallèle était meilleur que tous les résultats publiés au moment de l'expérimentation.

Le même article traite également d'une version séquentielle et parallèle du recuit simulé, une autre méthode pour contrer les optimum locaux. Les auteurs comparent le tabou et le recuit. Leurs résultats sont que le tabou est plus rapide et donne de meilleurs solutions que le recuit dans leurs versions séquentielles. Par contre les versions parallèles sont jugées

[7] Nous ne comprenons pas pourquoi ils n'ont pas utilisé 8 processeurs puisque la machine en possède 10.

à peu près équivalentes.

Les auteurs n'avancent directement aucun chiffre en ce qui concerne le "speedup" obtenu par la recherche avec tabous en parallèle. Nous avons une certaine idée de ce qu'ont dû être ces résultats en faisant un recoupage des graphiques présentés dans l'article; à notre avis ce n'est pas très reluisant. C'est tout à fait différent pour le recuit, les "speedup" sont intéressants et là les données directes abondent. Entre autre, les auteurs rapportent avoir obtenu des speedup de 11 sur expérimentation parallèle avec deux processeurs et un problème de taille 100. Il faut noter que des speedup non linéaires ne sont pas inhabituelles pour des recherches faites avec plusieurs processeurs dans un espace de solutions. Des speedup exponentiels dans le nombre de processeurs sont théoriquement et techniquement réalisables [26, 28, 33], il reste cependant à développer un modèle d'algorithmique parallèle capable de générer ces speedup sur une base constante.

8.5 Réseaux de neurones

Nous allons maintenir décrire une dernière application de la méthode des tabous concernant cette fois un problème d'optimisation relié au processus d'apprentissage dans les réseaux de neurones. Nous tirons principalement la description de cette application d'un article de De Werra et Hertz [4] et de certains articles concernant les réseaux de neurones [7, 32, 53]. Nous n'avons pas fait l'hypothèse que le lecteur était familier avec les réseaux de neurones, aussi dans un premier temps nous faisons une brève introduction à l'historique et au fonctionnement des réseaux formels de neurones.

8.5.1 Le modèle de réseaux de neurones

On attribue l'origine des travaux sur les réseaux de neurones aux articles de McCulloch et Pitts [3] et de Rosenblatt [55]. Au début, les réseaux de neurones avaient comme principale application la compréhension de phénomènes neurophysiologiques, puis ils ont évolués graduellement vers des applications qui concernent principalement les processus d'apprentissage par les machines dans la reconnaissance des formes ou la reconnaissance de symboles manuscrits. Aujourd'hui, on peut dire que les propriétés et le comportement des réseaux de neurones sont assez loin des préoccupations originales des vrais neurones de notre cerveau. Il y a trois raisons principales qui ont amené les chercheurs à se désintéresser de la modélisation des vrais neurones: 1- on ne connaît pas bien les vrais neurones, 2- même si on les connaissaient très bien, en termes d'efficacité de calcul, il serait probablement peu intéressant d'en implanter toutes les propriétés, 3- il est probable qu'il ne soit pas nécessaire d'utiliser toute la complexité des vrais neurones pour les fins de donner des capacités d'apprentissage aux machines. Pour toutes ces raisons, on a aujourd'hui ce qu'on appelle les *réseaux de neurones artificiels* ou encore *réseaux de neurones formels* pour les distinguer de modèles plus réalistes (par rapport aux vrais neurones).

Jusqu'en 1985, il y avait presqu'autant de types de réseaux de neurones que de chercheurs dans le domaine. Le modèle de Hopfield [32] a fini par s'imposer plus que les autres, de

telle sorte que la généralisation suivante des réseaux de neurones s'en inspire largement.

Un réseau de neurones formel est construit d'unités de base appelées neurones[8]. Chaque neurone est relié à d'autres neurones par des arcs qui correspondent aux synapses des vrais neurones. Pour chaque arc entre les neurones i, j est associé un poids c_{ij} correspondant à l'effet synaptique (qui inhibe ou excite le neurone récepteur). À chaque neurone correspond un état, qui est calculé en faisant la somme des inputs (qui sont fonction des poids des arcs entrant), en soustrayant une valeur seuil, et en appliquant une fonction d'activation du neurone. On a donc les principaux paramètres suivants:

t = 1, 2, ... = temps
$x_i(t)$ = état du neurone i
c_{ij} = poids du lien entre i et j
Θ_j = seuil
S = fct d'activation.

La transformation d'un neurone peut être caractérisée de la façon suivante [7]:

$$x_j(t+1) = S(\sum_i c_{ij} x_i(t) - \Theta_j)$$

Pour pouvoir exécuter leurs tâches, ces machines ont besoin en quelque sorte d'être programmées. Cette programmation se fait en donnant des valeurs aux paramètres c_{ij} et Θ_j à l'aide d'une sélection basée sur des règles d'apprentissage. Les règles classiques dans le domaine sont les règles de Hebbien, du neurophysiologiste D.O. Hebb de McGill. Le principe de ces règles est très simple: l'efficacité d'une synapse se renforce lorsque les neurones reliés par cette synapse sont actives ou passives en même temps, cette efficacité se détériore dans le cas contraire. On peut implanter ce principe en faisant varier le poids de chaque arc en fonction du produit de l'activité des neurones du lien: $\Delta c_{ij} = c(x_i \times x_j)$.

L'apprentissage des réseau de neurones est obtenu à l'aide d'un vecteur v soumis en input au réseau et à la variation des paramètres c_{ij} et Θ jusqu'à ce que l'ensemble des états des neurones se stabilise autour d'un vecteur (prototype) v'. On dit alors qu'on a calculé les efficacités synaptiques du système tel qu'il sera capable de reconnaître v' si un vecteur semblable à v' est soumis en input. La phase d'apprentissage est alors terminée. C'est cette phase d'apprentissage qui peut être réalisée à l'aide de la méthode des tabous.

8.5.2 Le modèle de réseau de neurones utilisé

L'application de la méthode des tabous que nous allons voir ici consiste à optimiser le calcul des efficacités synapsiques, c'est-à-dire qu'on veut que le système ait le moins d'états

[8] La simulation de ces neurones dans un ordinateur séquentiel se fait à l'aide d'un tableau de valeurs numériques, dans les systèmes parallèles la simulation se fait à l'aide d'un processeur d'une grande simplicité, dans le style de ceux de la Connection Machine [53].

parasites (nous reviendrons sur ce sujet plus loin) possible, et qu'il puisse reconnaître un prototype avec un nombre minimum d'itérations.

Le modèle de réseau de neurones adopté pour cette expérimentation est celui de [32]. Soit un ensemble de n neurones, où chaque neurone pour une itération donnée t est soit dans un état actif ou dans état passif. L'état de chaque neurone sera représenté par $\rho_i(t)$ qui peut prendre les valeurs +1 et -1. Chaque neurone est relié à tous les autres pour former un graphe orienté complet. Chaque arc entre les neurones i et j est caractérisé par un poids c_{ij}. À l'itération t, chaque neurone i se voit associé un potentiel $v_i(t)$ défini par

$$v_i(t) = \sum_{j=1}^{n} c_{ij}\rho_j(t).$$

L'état $\rho_i(t+\Delta t)$ du neurone i à l'itération $t+\Delta t$ est calculé à partir de $v_i(t)$ et du seuil Θ_i:

$$\rho_i(t+\Delta t) = \begin{cases} +1 \text{ si } v_i(t) > \Theta_i \\ -1 \text{ si } v_i(t) < \Theta_i \\ \rho_i(t) \text{ si } v_i(t) = \Theta_i \end{cases}$$

Pour chaque neurone i, $v_i(t+\Delta t)$ est calculé à partir des valeurs de $\rho_j(t+\Delta t)$. Le système atteint alors un état stable $\underline{\rho'} = (\rho'_1, ..., \rho'_n)$ après quelques itérations. L'état final $\underline{\rho'}$ est fonction de l'état initial $\underline{\rho}$ à l'itération $t = 0$ du réseau de neurones, de la matrice $C = c_{ij}$ et du seuil Θ_i. On considérera ici que $\Theta_i = 0$.

La matrice C contient sous une forme implicite le programme du réseau. Il s'agit de construire une matrice C qui mémorise certains états prototypes représentés par les vecteurs (prototypes) $\underline{\rho^1}, ..., \underline{\rho^p}$ ($\rho^i \in \{-1, +1\}^n$ pour $i = 1, ..., p$). Ces prototypes sont les vecteurs que le système doit pouvoir reconnaître lorsqu'un vecteur $\underline{\rho}$ est présenté au réseau de neurones[9].

Soit $H(\lambda, \mu)$ la distance entre deux états $\underline{\lambda}$ et $\underline{\mu}$ du réseau comme étant le nombre de composantes j où $\lambda_j \neq \mu_j$. On veut que lorsqu'un vecteur λ^k est présenté en input au réseau avec $H(\lambda^k, \mu^k) \leq d$, il stabilise à l'état $\underline{\mu^k}$, c'est-a-dire on veut imposer l'association (λ^k, μ^k). En général, il y aura une collection d'associations (λ^k, μ^k), pour $k = 1, ..., q$ que l'on voudra imposer au réseau. Pour un état $\underline{\rho}$ au temps t, on dénote par $\underline{\rho}(C)$ le nouvel état du réseau au temps $t + \Delta t$ calculé avec la matrice courante C.

8.5.3 Le problème d'apprentissage formulé comme un problème d'optimisation

On décrit ce problème d'apprentissage sous la forme d'un problème d'optimisation de la façon suivante [4]:

$$f(C) = \sum_{k=1}^{q} H(\underline{\lambda^k}(C), \underline{\mu^k}).$$

[9] C'est-à-dire que le système doit évoluer et se stabiliser après quelques itérations avec $\underline{\rho'}$ comme vecteur représentant l'état de chaque neurone du réseau.

Si $f(C) = 0$, alors l'association est réalisée dès la première itération. Le processus d'apprentissage du réseau de neurones consiste à trouver une matrice $C = (n \times n)$ qui minimise cette somme des distances. Il y aura donc n^2 coefficients c_{ij} à déterminer.

8.5.4 Définition des solutions voisines

L'ensemble X des solutions réalisables est formé de toutes les matrices $R^{n \times n}$. Soit $C[i,j;\epsilon]$ la matrice obtenue de C en ajoutant ϵ à c_{ij}. La valeur c_{ij}^+ sera le plus petit ϵ pour lequel il existe au moins une association de (λ^k, μ^k) où le vecteur produit par $\underline{\lambda^k}(C)$ est différent du vecteur de $\underline{\lambda^k}(C[i,j;\epsilon])$. De même c_{ij}^- est le plus petit ϵ tel que $\underline{\lambda^k}(C) \neq \underline{\lambda^k}(C[i,j;-\epsilon])$ pour un k donné. Si la solution courante x est la matrice C, ses solutions voisines $E(x)$ sont formées de toutes les matrices $C[i,j;c_{ij}^+], C[i,j;c_{ij}^-]$ $(i,j = 1,...,n)$ [10].

8.5.5 Liste candidate et évaluation de la meilleur transformation

La liste candidate est ici composée d'un sous-ensemble des transformations disponibles à chaque itération. Soit K^1 l'ensemble des associations pour lesquelles un accroissement de c_{ij}^+ de c_{ij} réduit la distance $H(\underline{\lambda^k}(C), \underline{\mu^k})$ de un. Soit K^0 l'ensemble des associations pour lesquelles la variation c_{ij}^+ augmente la distance de un. Pour chaque paire de neurones (i,j) le calcul suivant est fait [4]

$$P_{ij}^+ = -\sum_{k \in K^1} w_k + \sum_{l \in K^0} w_l$$

où w_k est un poids de valeur positive qui est donné à chaque association (λ^k, μ^k) relativement à son importance. P_{ij}^+ est le changement dans la fonction objectif $f(C)$ lorsqu'une correction c_{ij}^+ est faite à c_{ij}. On définit de façon similaire P_{ij}^- qui est le changement $f(C)$ lorsqu'une modification c_{ij}^- est faite à c_{ij}.

La transformation choisie pour appliquer à la solution courante est caractérisée par la paire de neurones (i,j) qui minimise (P_{ij}^+, P_{ij}^-). On choisit donc la transformation qui améliore le plus la fonction objectif.

8.5.6 Les listes tabous et critères d'aspiration

Il y a deux listes tabous T^+ et T^- qui contiennent les k dernières paires (i,j) pour lesquels c_{ij} était respectivement décroissant et croissant.

Il y a un seul critère d'aspiration, c'est la meilleur amélioration de la fonction objectif obtenue en une seule itération.

---

[10] Le lecteur pourra consulter l'étude de De Werra et Hertz [4] pour le détail du calcul de c_{ij}^+ et c_{ij}^-.

8.5.7 Fonctionnement de la recherche avec tabous

Une itération de la recherche avec tabous fonctionnera de la façon suivante. À partir d'une solution $x = C$, on choisi une paire (i, j) de neurones pour laquelle $min(P_{ij}^{+}, P_{ij}^{-})$. On vérifie si cette paire est contenue dans les listes tabous T^{+} ou T^{-}. Si la paire (i, j) est tabou, on vérifie le critère d'aspiration, si la variation de f est plus grande que celle contenue dans le critère d'aspiration, on accepte la transformation de C à C' et on met à jour le critère d'aspiration. Si (i, j) n'est pas tabou, la transformation est appliquée, les listes tabous sont mises à jour de même que le critère d'aspiration si nécessaire.

Le processus est répété jusqu'a ce qu'aucune amélioration de f n'est été obtenue depuis un certain nombre d'itérations ou jusqu'à ce que $f = 0$.

8.5.8 Tests et résultats

Il semble que les auteurs de l'article n'aient fait eux-mêmes qu'un seul des trois tests de l'étude, rapportant les résultats de 2 tests réalisés ailleurs. Un premier test est celui du réseau de 8 neurones de Personnaz, Guyon et Dreyfus [53]. Dans cette expérience, il y avait 5 états prototypes et 36 associations possibles avec $H(\Delta^{k}(i), \varrho^{k}) = 1$ pour $(i = 1, ..8)$. Une matrice C a été construite telle dans tous les cas où le vecteur prototype était soumis au réseau (c'est-à-dire l'association du vecteur prototype avec lui-même), il était immédiatement reconnu. Des 36 associations possibles le réseau avait été capable d'en réaliser 34 en une seule itération (évidemment les beaux cas).

Un des problèmes pour les réseaux de neurones vient des faux états attracteurs, c'est-à-dire ces états où la recherche stabilise mais qui ne correspondent à aucun des vecteurs prototypes que l'on voudrait que le système puisse reconnaître[11]. Une bonne fonction d'apprentissage doit tendre à faire en sorte que la matrice C minimise ces états parasites. La matrice C obtenue de la recherche avec tabous plus haut avait 13 états stables, c'est-à-dire les 5 états prototypes, les 8 autres correspondaient à des états parasites. Les auteurs mentionnent que la matrice C construite avec la recherche avec tabous réalisait plus d'associations qu'une matrice C construite avec une autre méthode, mais ils ne font pas mention du résultat de la comparaison au niveau des états parasites.

Une autre expérimentation avec la recherche avec tabous est rapportée, cette fois le réseau était formé de 20 neurones, il y avait 2 prototypes et 50 associations pour chaque prototype. Sur environ un million d'états possibles, il y en avait 4 qui étaient stables, c'est-à-dire les deux prototypes et deux états parasites. Les états parasites étaient à une distance H d'au moins 5 des prototypes. La conclusion de cette expérimentation fut que le nombre d'états parasites reste faible si le nombre de prototypes est plus petit ou égal à .15 fois le nombre de neurones.

L'expérimentation réalisé par les auteurs a été faite avec deux prototypes et un réseau

[11] Selon la fameuse analogie avec les niveaux d'énergie minimum d'un système (qu'utilisent aussi les adeptes du recuit simulé), l'apprentissage revient à creuser des trous aux bons endroits dans le paysage d'énergie du réseau.

de 25 neurones. Chaque prototype avait 50 associations $(\lambda^k(i), \varrho^k)$ avec $H(\lambda^k(i), \varrho^k) \leq 6$. Après 3584 itérations de la recherche avec tabous, ils avaient obtenu une matrice C capable de réaliser toutes les associations en moins de 3 itérations.

9 Conclusion

Glover a introduit la recherche avec tabous dans un article dont le titre fait le lien entre la programmation en nombre entier (R.O.) et l'intelligence artificielle. La recherche avec tabous telle que décrite dans cet article correspond d'assez près à la description que nous avons fait de cette méthode dans la section 4. C'est dans une autre section de l'article où Glover traite de méthodes de résolution de problèmes combinatoires basée sur des stratégies d'apprentissage. Il y est question entre autre de l'analyse par objectifs et de méthodes de décomposition basée sur des principes de l'intelligence artificielle. C'est uniquement dans plusieurs articles récents où Glover intègre les stratégies d'apprentissage et la recherche avec tabous.

Ce rapprochement entre l'intelligence artificielle et la recherche opérationnelle pour résoudre des problèmes combinatoires représente à notre avis une tentative à la fois contestable et prometteuse.

Cette tentative est contestable parce qu'on pourrait se demander comment démontrer mathématiquement vers quelle solution cette méthode converge. Comme nous l'avons expliqué dans l'introduction de ce travail, la forme d'apprentissage dont il est question pour la recherche avec tabous est basée sur la mémoire. Tout le monde sait que les mathématiques sont incapables d'expliquer le comportement intelligent. Mais malheureusement ce n'est pas leur seule limitation.

Comme le mentionne Glover lui-même, l'état actuel des mathématiques ne permet pas d'expliquer le comportement de tout système qui inclut une dimension de mémoire ou de temps. Les théoriciens des langages de programmation le savent très bien, les changements du contenu de la mémoire par (:=) leur causent bien des mots de tête. Les mathématiques sont généralement reconnues pour traiter le temps de façon effective seulement au niveau des équations différentielles. Tous ceux qui sont familier avec le calcul parallèle en connaissent les conséquences[12]. Étant donné cette limitation, on voit alors mal comment on pourrait démontrer quelques propriétés de convergence que se soit pour une méthode qui repose largement sur l'historique de la recherche pour décider de sa progression.

Mais on peut se demander s'il est si important de démontrer mathématiquement l'efficacité d'une méthode. Par exemple, pour le recuit simulé on peut démontrer que cette méthode converge vers une solution optimale en un nombre infini d'itérations! Est-ce vraiment important pour les problèmes concrets à solutionner?

Plus fondamentalement, ce qui nous semble avoir conduit Glover à lier intelligence

[12] C'est une des raisons pour lesquelles les mathématiques n'arrivent pas à modéliser le comportement d'un système parallèle. Le formalisme du calcul et des équations différentielles assume une seule dimension de temps. Or la multiplicité et la relativité des espace/temps de chaque processus concurrent est incompatible avec cette hypothèse.

artificielle et recherche opérationnelle est une sorte de position épistémologique qui consiste à penser que les progrès en mathématique ne sont pas nécessairement garant de progrès au niveau du calcul informatique. Nous citons ici un extrait des motivations qui soutienne cette position au niveau des problèmes combinatoires.

> ... a procedure with special convergence properties, demonstrable by theorem and proof, must be somewhat rigidly structured, and this must impose limitations in turn on performance... flexibility can be the enemy of theory,...yet rigidity can be equally the enemy of effective search. In the face of combinatorial complexity, there must be freedom to depart from the narrow track that logic single-mindedly pursues,...effective strategies for combinatorial problems can require methods that formal theorems are unable to justify. [14]

Nous pensons que le même genre d'observations pourraient être fait au niveau de l'exploitation du parallélisme pour la méthode des tabous.

Dans un article récent de plus d'une centaine de pages, Pardalos, Phillips et Rosen [52] font une synthèse de tous les travaux sur la résolution de problèmes combinatoires en parallèle. Dans leur introduction, les auteurs listent toutes les formes de parallélisme exploitables pour cette catégorie de problèmes: décomposition du programme (ou du problème) en deux ou plusieurs tâches telle que chacune d'entre elles puisse être exécutée sur un processeur différent. Et voilà, c'est tout, pas d'autre forme exploitable de parallélisme.

C'est ce genre d'énoncé qui a conduit une physicienne[13] à faire l'observation selon laquelle les informaticiens manquaient complètement d'imagination pour le design d'algorithmes parallèles et à faire l'hypothèse que probablement ces derniers n'y comprenaient pas grand chose à l'intérêt réel de faire du calcul parallèle. Manthey [48] et Crowley [2] pointent un peu dans la même direction, de même que Kaehr et von Goldammer [36] lorsqu'ils disent que le modèle de parallélisme actuel en informatique est incapable de simuler la plupart des découvertes récentes en neurophysiologie par exemple.

Ce qui exaspère ces gens, c'est que les informaticiens ont transposé au niveau du calcul parallèle l'approche réductionniste qui consiste à penser qu'une procédure n'est jamais plus que l'ensemble de ses fonctions. Selon cette approche, ce que peut calculer une machine avec 10 processeurs ne sera jamais plus que ce que peut calculer une machine séquentielle en 10 fois plus de temps. La franchise de plusieurs scientifiques vient de ce qu'ils croient voir dans les machines parallèles un potentiel de calcul bien supérieur à ce que y voient les informaticiens. Pour ces scientifiques, les machines parallèles offrent la possibilité de simuler plusieurs phénomènes en physique, chimie ou biologie qui ne peuvent pas l'être sur une machine séquentielle.

Nous sommes en général tout à fait d'accord avec l'opinion de ces chercheurs en sciences naturelles. Il nous semble que cette critique n'est pas tout à fait étrangère à celle de Glover. Bien que le paradigme de parallélisme que Glover suggère pour sa méthode soit tout à fait le paradigme classique de parallélisme, nous croyions qu'il serait possible d'appliquer au

[13] C'était à l'une des conférences les plus prestigieuse sur le parallélisme, ICPP 1988 à Chicago, devant plus de 200 des informaticiens qui se croyaient à la fine pointe du calcul parallèle.

tabou le paradigme de parallélisme d'émergence auquel font référence plusieurs physiciens, chimistes et biologistes.

L'intelligence est probablement un des phénomènes émergents les plus importants. Il serait possible de simuler au niveau de la mémoire à long terme de la recherche avec tabous, certaines propriétés émergentes de l'intelligence dans le but de faire de la recherche dans un espace complexe de solutions. Bien que nous n'ayons pas une idée précise comment cela pourrait se faire, il n'en demeure pas moins que la grande flexibilité de la recherche avec tabous laisse place plus que toute autre méthode à ces formes d'exploitation du parallélisme. Les connectionnismes sont probablement le germe d'une véritable algorithmique parallèle, les développements qui auront lieu de ce côté couplé avec l'identification de certains phénomènes émergents de l'intelligence humaine pourront aider à avoir une meilleurs idée pour appliquer ce parallélisme à la méthode avec tabous.

Remerciements

Nous tenons à remercier les professeurs Gilles Brassard et Theodor Crainic pour avoir fait la révision de ce texte et y avoir suggéré de nombreuses améliorations.

References

[1] J. Bovet and D. De Werra, A convoy scheduling problem, *Discrete Applied Mathematics*, 1990.

[2] C.S. Crowley, Structured programming is reductionistic, *SigPlan Notices*, 15, 1980.

[3] W.S. McCulloch et W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.*, 5(1943), 115–133.

[4] D. de Werra et A. Hertz, Tabu search techniques: a tutorial and an application to neural networks, *OR Spectrum 11*, 1989, 131–141.

[5] B.T. Eck, Good solutions to job scheduling problems via tabu search, Research Report, Department of Industrial Engineering & Operations Research, Columbia University, New York, 1989.

[6] U. Faigle and W. Kern, Some convergence results for probabilistic tabu search, Research Memorandum No. 882, Faculty of Applied Mathematics, University of Twente, The Netherlands, 1989.

[7] J.D. Farmer, A rosetta stone for connectionism, *Physica*, D 42(1990), 153–187.

[8] J.D. Farmer, S.A. Kauffman and N.H. Packard, Autocatalytic replication of polymers, *Physica*, D22, 1986, 50–67.

[9] M.S. Fox, Constraint-directed search: A case study of job-scheduling, Ph.D. Thesis, Carnegie-Mellon University, 1983.

[10] J. Frendewey, Candidate list strategies for improved implementation of the simplex/SON method, Ph.D. Thesis, School of Business, University of Colorado, Boulder, 1983.

[11] C. Friden, A. Hertz and D.De Werra, TABARIS: An exact algorithm based on tabu search for finding a maximum independent set in a graph, *Swiss Federal Institute of Technology*, Lausane, 1989.

[12] C. Friden, A. Hertz and D.De Werra, Stabulus: A technique for finding stable sets in large graphs with tabu search, *Computing*, 1990.

[13] F. Glover, Candidate list strategies for GN and simplex SON method, *Management Science Report Series*, MSRS 83-9, University of Colorado, Boulder, 1983.

[14] F. Glover, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, vol. 13, no.5, (1986), 533–549.

[15] F. Glover, Tabu search, Part I, *ORSA Journal of Computing*, vol. 1, no.3, (1989), 190–206.

[16] F. Glover, Candidate list strategies and tabu search, *CAAI Research Report*, University of Colorado, Boulder, 1989.

[17] F. Glover, Artificial intelligence, heuristic frameworks and tabu search, *Manuscrit.*

[18] F. Glover, Tabu search: A tutorial, *Interfaces*, 20:4, (1990), 74–94.

[19] F. Glover, Tabu search, Part II, *ORSA Journal of Computing*, vol. 2, no.1, (1990), 4–32.

[20] F. Glover, Issues and methods for applying target analysis to tabu search, *Manuscrit*

[21] F. Glover, Tabu search for the graph partitioning problem: illustrative organization and the cluster move strategy, *Manuscrit.*

[22] F. Glover and C. McMillan, The general employee scheduling problem: An integration of management science and artificial intelligence, *Computers and Operations Research*, vol. 13, no. 5, 1986, 563–593.

[23] F. Glover, M. Laguna, Target analysis to improve a Tabu search method for machine scheduling *Technical Report*, Advanced Knowledge Research Group, US West Advanced Technologies, Boulder, Colorado, 1989.

[24] F. Glover, H.J. Greenberg, New approaches for heuristic search: A bilateral linkage with artificial intelligence, *European Journal of Operational Research*, vol. 39, no.2, (1989), 119–130.

[25] F. Glover, D. Klingman and N. V. Phillips, A network-related nuclear power plant model with an intelligent Branch-and-Bound solution approach, *Annals of OR*, **21**(1989), 317–332.

[26] J.L. Gustafson, Fixed time, tiered memory, and superlinear speedup, *Proc. of the Fifth Conf. on Distributed Memory Computers*, 1990.

[27] P. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem, *RUTCOR Research Report*, Rutgers University, New Brunswick, N.J., 1987, 43–87.

[28] D.P. Helmbold and C.E. McDowell, Modeling speedup(n) greater than n, *1989 International Conference on Parallel Processing Proceedings*, vol. 3, 1989, 219–225.

[29] A. Hertz and D.De Werra, Using tabu search techniques for graph coloring, *Computing*, vol. 29, 1987, 345–351.

[30] A. Hertz and D.De Werra, The tabu search metaheuristic: How we used it, *Annals of Mathematics and Artificial Intelligence*, 1990.

[31] J. H. Holland, Adaptation in natural and artificial systems, *The University of Michigan Press*, Ann Arbor, 1975 (livre).

[32] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences*, USA, **79**(1982), 2554–2558.

[33] B.A. Huberman, The performance of cooperative processes, *Physica* **D42**, 1990, 38–47.

[34] B. Jaumard, P. Hansen and Poggi de Aragao, Column generation methods for probabilistic logic, *Gerad*, McGill University, 1989.

[35] N.K. Jerne, Towards a network theory of the immune system, *Ann. Immunology*, Inst. Pasteur, 125 C, 1974, 373–389.

[36] , R. Kaehr and E. von Goldammer, Again computers and Brain, *Journal of molecular electronics*, vol. 4, 1988, S31–S37.

[37] J.O. Kephart, T. Hogg and B.A. Huberman, Dynamics of computational ecosystems: Implication for DAI, *Distributed Artificial Intellignece*, vol. 2, Kaufmann, Los Altos, CA, 1989.

[38] J.O. Kephart, T. Hogg and B.A. Huberman, Collective behavior of predicitive agents, Physica D, 1990 48–65.

[39] S. Kirkpatrick, C. D. Jr. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science*, Vol. 220, # 4598, 1983, 671–680.

[40] J. Knox, The application of tabu search to the symmetric traveling salesman problem, Ph.D. Thesis, Graduate School of Business, University of Colorado, July 1989.

[41] J. Knox and F. Glover, Comparative testing of traveling salesman heuristics derived from tabu search, genetic algorithms and simulated annealing, *Center for Applied Artificial Intelligence*, University of Colorado, 1989.

[42] M. Laguna, Tabu search methods for a single machine scheduling problem, *Technical Report*, Advanced Knowledge Research Group, US West Advanced Technologies, Boulder, Colorado, 1989.

[43] M. Laguna, A tabu search method for scheduling jobs on parallel processors, *Research Report*, Department of Mechanical Engineering, The University of Texas-Austin, 1989.

[44] M. Laguna, J.W. Barnes and F. Glover, Scheduling jobs with linear delay penalties and sequence dependent setup costs using tabu search, *Research Report*, Department of Mechanical Engineering, University of Texas-Austin, April 1989.

[45] M. Lee, Least-cost network topology design for a new service using tabu search, *Heuristics for Combinatorial Optimization*, Section 6, 1989, 1–18.

[46] M. Malek, M. Guruswamy, H. Owens and M. Pandya, Serial and parallel search technique for the traveling salesman problem, *Annals of OR: Linkages with Artificial Intelligence*, 21(1989), 59–84.

[47] M. Malek, M. Heap, R. Kapur and A. Mourad, A fault tolerant implementation of the traveling salesman problem, *Research Report*, Department of Electrical and Computer Engineering, University of Texas-Austin, 1989.

[48] M.J. Manthey, Hierarchy in sequential and concurrent systems, Computing Research Laboratory, New Mexico State University, 1988.

[49] R.A. Maxion, Toward diagnosis as an emergent behavior in a network ecosystem, Physica D, 1990, 67–83.

[50] S. Oliveira and G. Stroud, A parallel version of tabu search and the path assignment, *Heurisitcs for Combinatorial Optimization*, Section 4, 1989, 1–24.

[51] P.S. Ow and T.E. Morton, Filtered beam search in scheduling, *Int. J. Prod. Res*, vol. 26, no. 1, 1988, 35–62.

[52] P.M. Pardalos, A.T. Phillips and J.B. Rosen, Topics in parallel computing in mathematical programming, *Manuscrit*

[53] L. Personnaz, I. Guyon et G. Dreyfus, Collective computational properties of neural networks: new learning mechanisms, *Physical Review*, A34, (1986), 4217–4227.

[54] C.C. Pettey. M.R. Leuze and J.J. Grefenstette, Genetic algorithms on a hypercube multiprocessor, 1987.

[55] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Rev.* 65(1958), 386.

[56] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, *ORSA Journal of computing*, vol. 2, no. 1, 1989, 33–45.

[57] C. Wendelin, Graph partitionning with the aid of the tabu method, *Project Report*, Institute fur Hohere Studien, Vienna, Austria, 1988.

[58] M. Windmer and A. Hertz, A new approach for solving the flowshop sequencing problem, *European Journal of Operational research*, 1990.